No. FHWA-RD-76-66

# LE DETECTION--PHASE III
# PASSIVE BUS DETECTOR/INTERSECTION PRIORITY SYSTEM DEVELOPMENT

## Vol. I. Project Overview and Technical Discussion

December 1975

Final Report

This document is available to the public
through the National Technical Information
Service, Springfield, Virginia 22161

Prepared for

FEDERAL HIGHWAY ADMINISTRATION
Offices of Research & Development
Washington, D.C. 20590

## NOTICE

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the object of this report.

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof.

Technical Report Documentation Page

|  |  |  |
|---|---|---|
| 1. Report No.<br>FHWA-RD- 76-66 | 2. Government Accession No. | 3. Recipient's Catalog No. |

4. Title and Subtitle
VEHICLE DETECTION - PHASE III
PASSIVE BUS DETECTOR/INTERSECTION PRIORITY SYSTEM DEVELOPMENT
Volume I - Project Overview and Technical Discussion

5. Report Date
December 1975

6. Performing Organization Code

7. Author(s)
P. Anderson, R. Larson, R. Lubke,
G. Putnam and D. Wick

8. Performing Organization Report No.
2551-40274

9. Performing Organization Name and Address
Honeywell Inc.
2600 Ridgway Parkway
Minneapolis, MN. 55413

10. Work Unit No. (TRAIS)
FCP-32L1012

11. Contract or Grant No.
DOT-FH-11-8149

12. Sponsoring Agency Name and Address
Federal Highway Administration
Office of Research and Development
Washington, D.C. 20590

13. Type of Report and Period Covered
Final Report

14. Sponsoring Agency Code
I-0181

15. Supplementary Notes
1. Contract Manager. Frank J. Mammano
2. National Semiconductor Manual, "IMP-16C Application Manual," is printed in its entirety as part of this report with permission of the copyright owner.

16. Abstract

The Passive Bus Detector/Intersection Priority System was developed under this project. The system functions as a fully independent traffic controller with completely passive bus detection capability using inductive-loop detectors (ILD) placed in the roadway. Bus detection is based on ILD signal processing (classifier) techniques, thereby eliminating need for any modifications or additions to the bus. Passive bus detection hardware includes special ILD electronics, a high-speed programmable processor, and a microprocessor. Using the same microprocessor, plus programming panel, stored programs, and switching and safety units, the system design is sufficient to exercise bus priority control at urban intersections containing up to 23 signal circuits and 16 ILD transducers. In addition, the system has capability of commanding the traffic signals in either one of three fixed timing patterns, selected at the system by the traffic engineer or controlled by a loop master.

17. Key Words
Passive Bus Detector, ILD
Signal Processing, Microprocessor,
Fully Independent Controller

18. Distribution Statement
Document is available to the public through the National Technical Information Service, Springfield, Virginia 22161

|  |  |  |  |
|---|---|---|---|
| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of Pages | 22. Price |
| UNCLASSIFIED | UNCLASSIFIED | 371 |  |

# TABLE OF CONTENTS

## TABLE OF CONTENTS (CONCLUDED)

## LIST OF ILLUSTRATIONS

## LIST OF ILLUSTRATIONS (CONTINUED)

## LIST OF ILLUSTRATIONS (CONCLUDED)

## LIST OF TABLES

# SECTION I
# INTRODUCTION

This final report covers Phase III of the vehicle detection project. The vehicle detection project, contract number DOT-FH-11-8149, is a contract between Honeywell Inc. and the Federal Highway Administration. The Phase III portion of this contract is entitled "Passive Bus Detector/Intersection Priority System" and was conducted from 1 July 1974 through 6 December 1975. E. H. Schmidt was project manager.

Other key personnel assigned to the Phase III portion of this project were:

| | | |
|---|---|---|
| D. O. Wick | - | Principal Investigator |
| P. A. Anderson | - | Systems Project Engineer |
| R. A. Lubke | - | Principal Development Engineer |
| R. M. Larson | - | Principal Research Scientist |
| D. G. Putnam | - | Senior System Scientist |
| R. M. Bonnema | - | Principal Development Engineer |

The following section contains a summary of project activities. Details on the project objectives and scope, technical approach and results, and conclusions and recommendations then follow.

# SECTION II
# SUMMARY

This report presents the results of a technology development project for the development and laboratory feasibility demonstration of an advanced concept called a Passive Bus Detector/Intersection Priority System.

The objectives of this system are twofold: (1) to accomplish passive detection of urban transit buses, and (2) to provide bus preemption through the intersection using traffic control strategies programmed into an independent microprocessor intersection controller. Under the Phase III project, two engineering models of a system were designed and built. Because the project objectives were to demonstrate system feasibility by means of engineering models rather than prototype models, adherence to NEMA or other design specifications were not contractually required and system components were designed for proof of performance rather than small size. The two systems built under the Phase III project will be field tested and evaluated in Minneapolis, Minnesota and Washington, D. C. under a separate contract modification.

As illustrated in Figure 1, the system is microprocessor based and is passive in the sense that neither bus driver action nor on-board equipment is required. Standard loop detectors are used for bus detection. Bus priority is accomplished by traffic signal plan modification through microprocessor software.

## BACKGROUND

In recent years, interest in bus priority has increased significantly as a means to increase bus ridership. Exclusive bus lanes to accomplish bus priority have been extensively evaluated and implemented both in this country and Europe. Techniques for providing bus priority through traffic signal control generally divides into two categories:

- Traffic signal preemption or modification in real-time using selective vehicle detectors

- Fixed time signal plans developed off-line (e. g. , using TRANSYT) providing bus preference.

Compared to exclusive bus lane operation, relatively little attention has been directed toward traffic signal control as a bus priority method. The most significant project addressing this problem in the United States is the Urban Traffic Control System/Bus Priority System (UTCS/BPS) research project being conducted by the FHWA. The objective of the BPS was to prove the

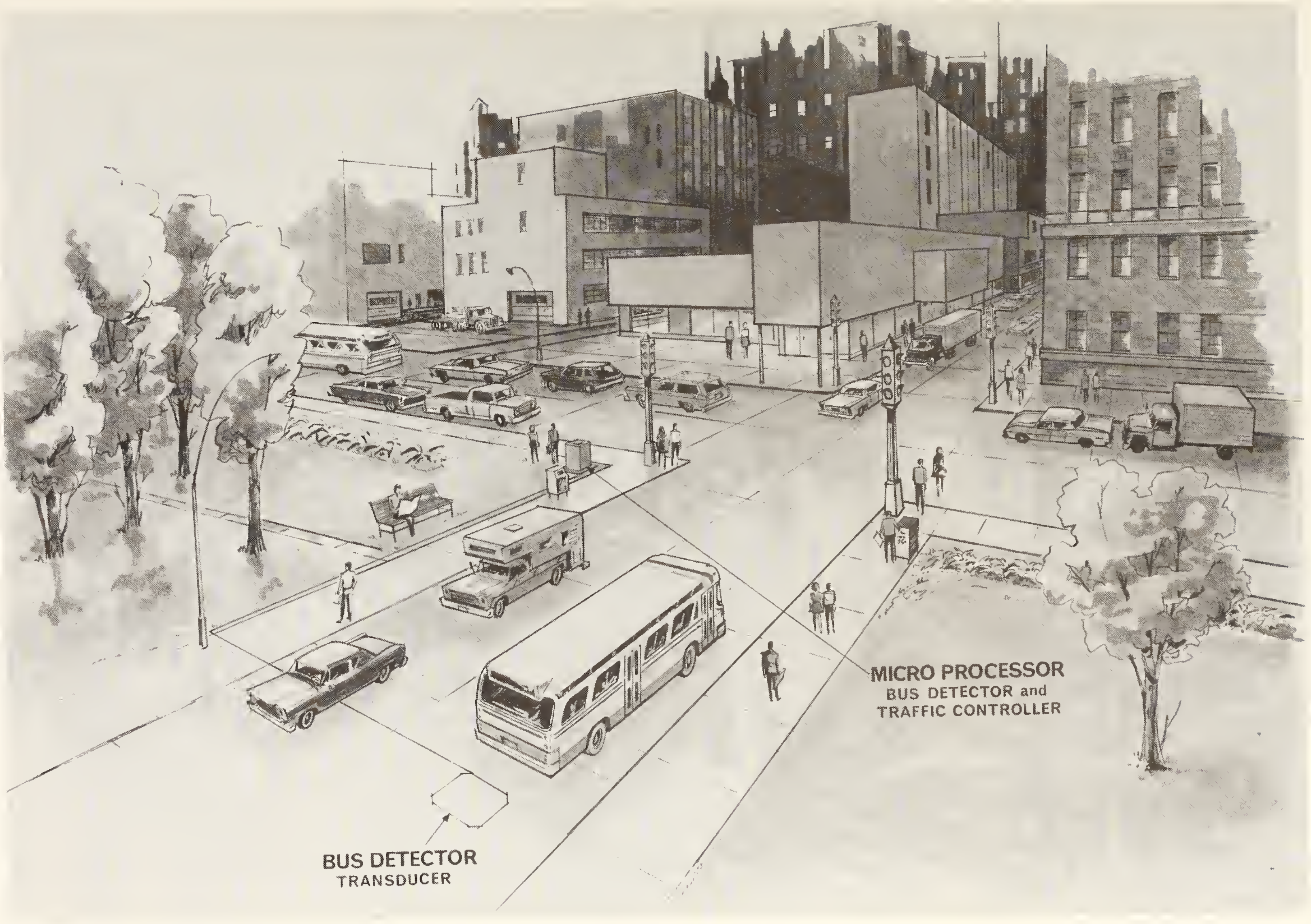


Figure 1. Passive Bus Detector/Intersection Priority System

flexibility and develop operational hardware and software for reducing bus delays in an urban network by providing preferential treatment to bus at signalized intersections.[1] After an investigation of available bus detector techniques, an active system employing a near-field radio technique was selected.[2] The BPS project demonstrated that the concept of reducing bus delays by providing preferential treatment at intersections is definitely feasible. Further, the hardware developed performed according to expectations. However, the requirement for on-board hardware and a special detector hinders the attractiveness of the BPS concept and prompted FHWA to consider passive bus detection approaches. The vehicle detection project, Phase III, was selected by the FHWA as the means to develop the passive approach.

## SUMMARY DESCRIPTION

Each system was developed to function as a fully independent intersection controller with passive bus detection capability using inductive loop transducers (ILD) buried in the roadways. For the Phase III project, each system bus priority strategy software is based on specific locations for the ILD. Changes in ILD locations will require modifications to the bus priority and, possibly, traffic control software. Further development in the prototype model(s) will examine techniques for rapid and simple preempt and control software changes that can be accomplished at the intersection or at the city traffic engineering facility. In the current two engineering models, each system is sufficient to exercise bus priority control at a specific intersection (Washington, D. C., and Minneapolis, MN) continuing up to 18 signal lights and up to four ILD units. Each system has built-in capability of being expanded to accept up to 16 ILD units and control of 23 signal circuits. The system capabilities are accomplished using microprocessor based electronics, a built-in programming panel, stored programs (software), and switching and safety units.

In addition, each system has capability of commanding the traffic signals in either one of three fixed timing patterns; selected at the system by the traffic engineer or controlled by the remote loop-master.

[1] Paper presented to ITE meeting, Seattle, Washington, August 1975. "Bus Priority and Bus Preemption in the Urban Traffic Control System", C. John MacGowan.

[2] PB 190847, "Advanced Control Technology in Urban Traffic Control Systems, Vol. IA, Sperry-Rand Corporation, March 1970.

A flow diagram of each system is depicted by Figure 2.

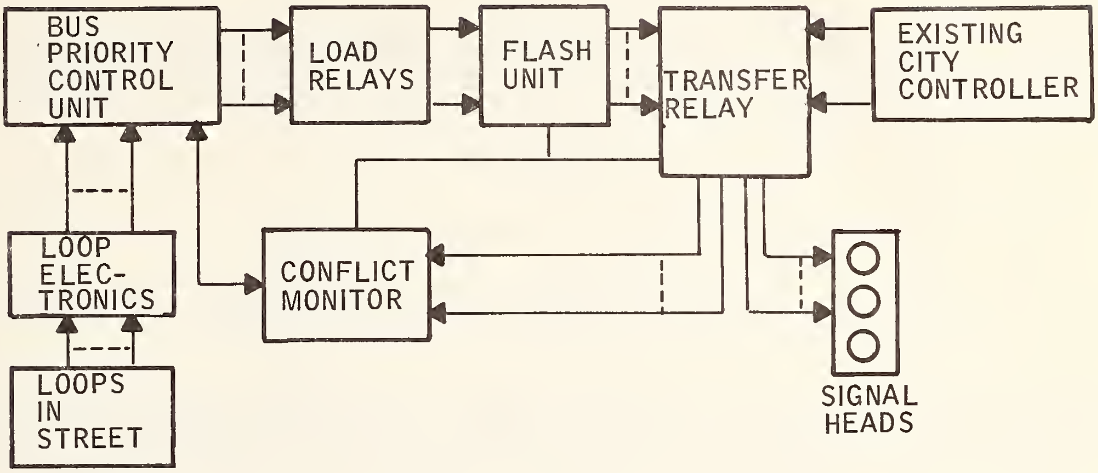


Figure 2. System Flow Diagram

The Loop Electronics and Bus Priority Control Unit represents the essential components of the system requiring extensive development in this project. The Bus Priority Control Unit outputs to the Load Relays and input/output to and from the Conflict Monitor represents an approximate dividing point where the existing system is significantly different from existing microprocessor controllers.

The inductive loop transducers in the street, combined with the specially developed loop electronics, will generate a time history signature of vehicles as they pass over the inductive loop transducer. A 1972 Flxible, 40-ft, urban transit bus has a signature as depicted in Figure 3.

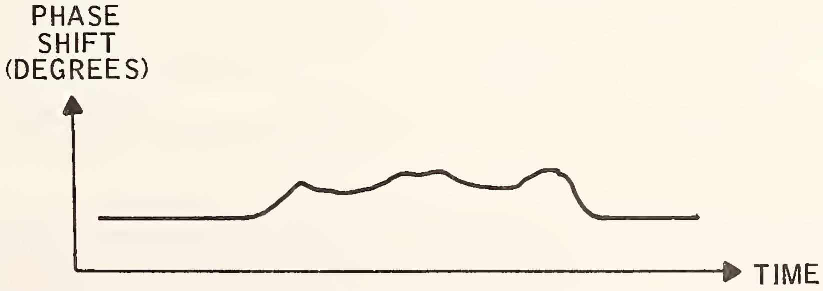


Figure 3. Bus Signature Derived from Inductive Loop

Other vehicles which are present in the urban traffic scenario will have time history signatures as depicted in Figure 4.

A cursory examination of the time history signatures depicted in Figures 4 and 5 illustrates the basic differences between the bus and all other vehicles. An analysis resolves the primary features of the bus:

a) Three distinct amplitude peaks

b) A distinct amplitude ratio between the peaks

c) A distinct location of each amplitude peak which is a function of their location with respect to vehicle length

The vehicle signatures are analyzed by the classification algorithm in the microcomputer unit of the Bus Priority Control Unit (Figure 2). The vehicle signatures are classified as a preempt bus or not. For a bus classification, the intersection traffic control software modifies the signal timing plan to favor the bus.

The microcomputer software performs bus classification, preemption and intersection traffic signal control. Modification of the timing plan for bus preemption is accomplished by moving time from contractible intervals to the extendible interval. The intervals in the timing plan are indicated by data for each detectorized preemptive approach to the intersection. This modification of the timing plan does not require any information about the cycle length, split or offset. Pedestrian clearances are not affected by the bus preemption. The timing plan always returns to synchronization on the cycle subsequent to the final bus preemption request.

The software functions of the system are summarized by Figure 6. Following the initialization sequence in which the real-time clock is started, the background computation cycle is entered. This cycle is repeated until a failure is detected. The periodic real-time clock interrupt initiates the real-time computations and then control is returned to the background cycle.

The electronic organization of the Bus Priority Control Unit is summarized by Figure 7. Figure 8 is a photograph of the Bus Priority Control Unit.

The Bus Priority Control Unit consists of five primary electronic components; described as follows:

a) Preprocessor (pP) - The preprocessor is a 12-bit programmable digital electronic unit including multiplex switches and an analog-to-digital converter capable of handling up to 16 inputs from inductive loop electronics having vehicle time-history signature extraction capability. The preprocessor

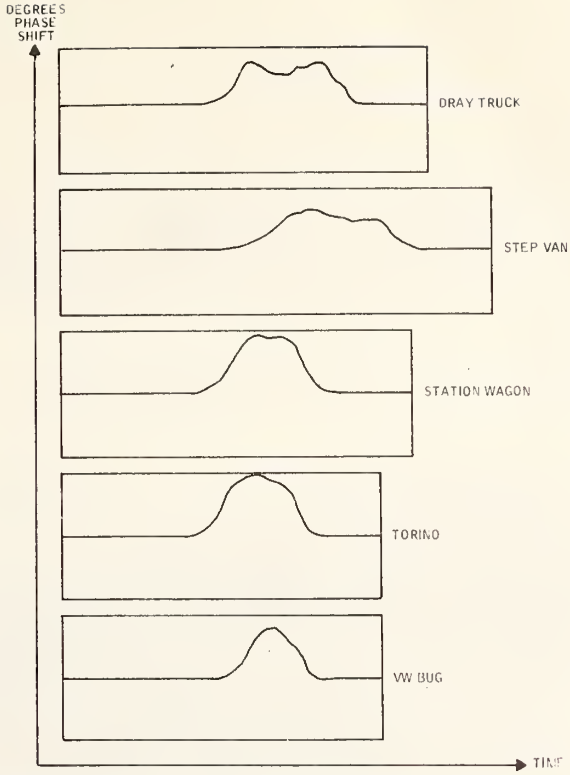
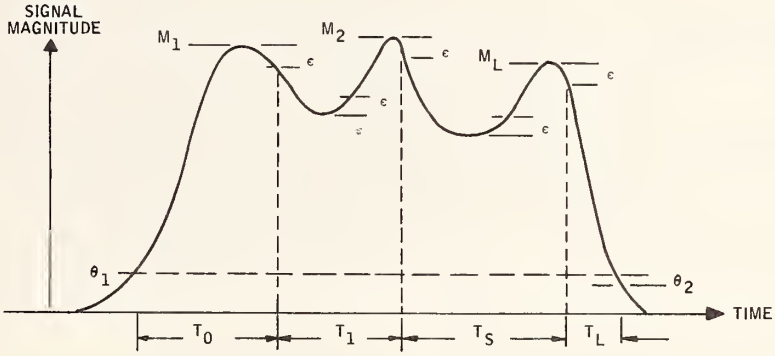


Figure 4. Examples of Vehicle Signatures Derived from Inductive Loop

These primary features are illustrated in Figure 5 using an idealized bus signature.



PRIMARY FEATURES

$T_0$ = TIME FROM TURN ON TO FIRST PEAK

$T_1$ = TIME FROM FIRST PEAK TO SECOND PEAK

$T_S$ = TIME FROM SECOND PEAK TO LAST PEAK

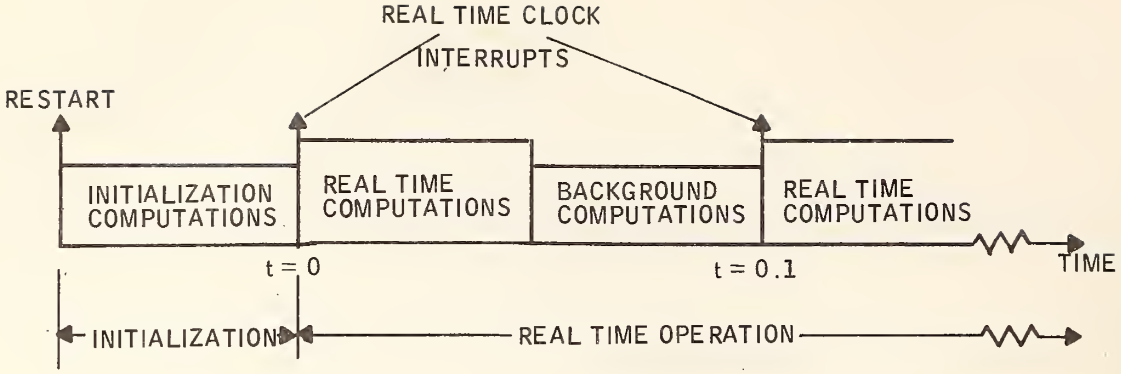
$T_L$ = TIME FROM LAST PEAK TO TURN OFF

$M_1$ = MAGNITUDE OF FIRST PEAK

$M_2$ = MAGNITUDE OF SECOND PEAK

$M_L$ = MAGNITUDE OF LAST PEAK

Figure 5. Bus Signature Primary Features



INITIALIZATION:
- INITIALIZE CPU STACK
- PERFORM SELF-TESTS
- INITIALIZE READ / WRITE MEMORY
- INITIALIZE TIMING PLAN EXECUTION
- RESET CONFLICT MONITOR
- START RTC
- START BACKGROUND COMPUTATIONS

REAL TIME:
- DATA OUTPUT
- DATA INPUT
- EXECUTIVE MONITORING, TIMING, SCHEDULING
- BUS PREEMPT (VEHICLE CLASSIFICATION, PREEMPT REQUEST, TIMING PLAN MODIFICATION)
- TIMING PLAN EXECUTION

BACKGROUND:
- SELF-TESTS

Figure 6. System Software Function Summary

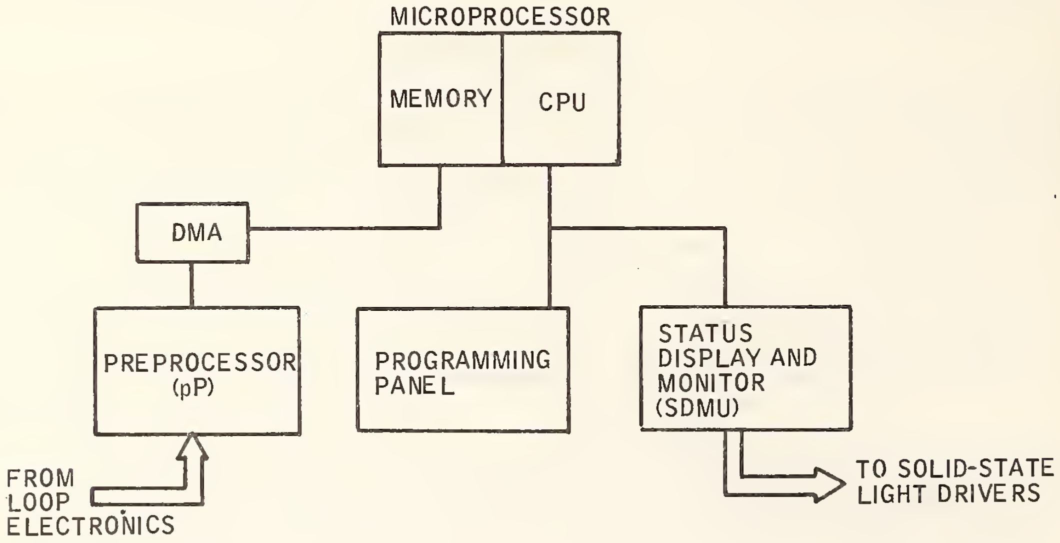


Figure 7. Bus Priority Control Unit Organization

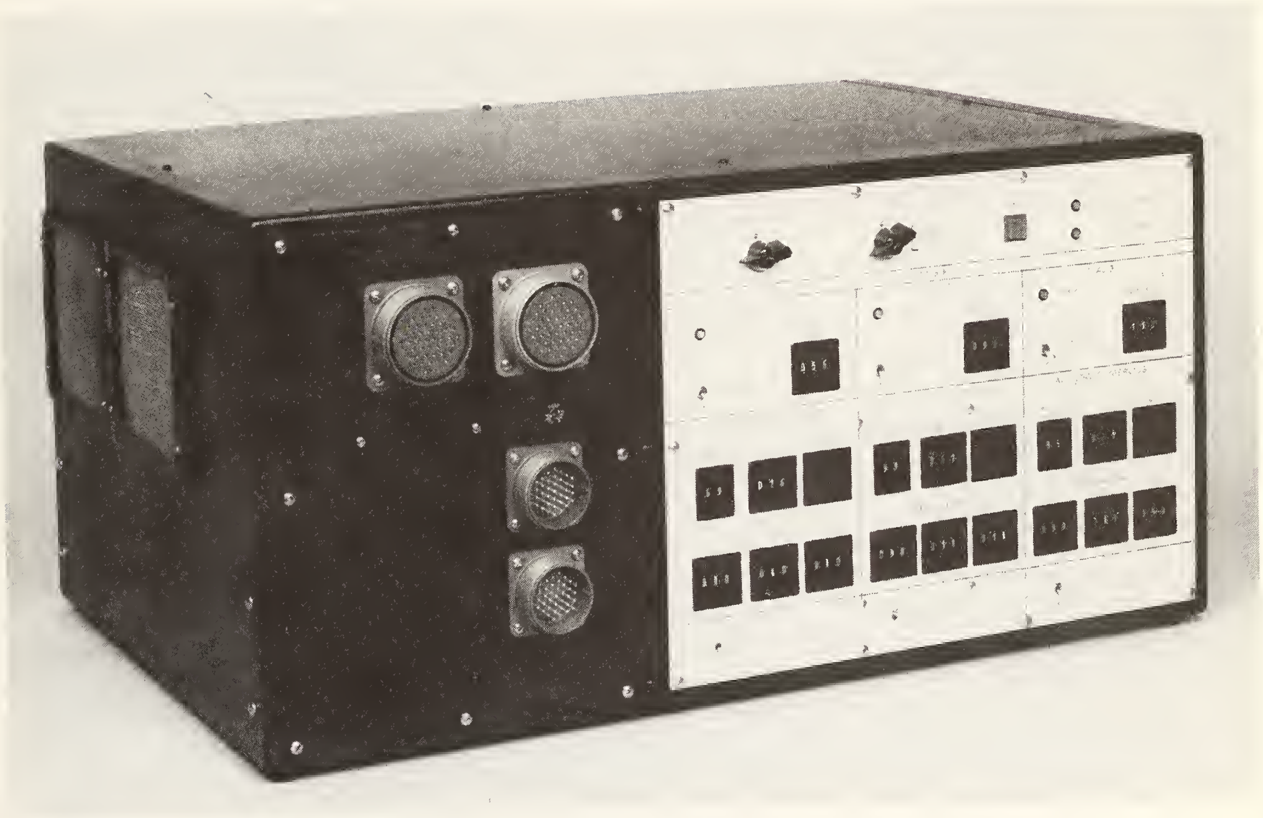


Figure 8. Bus Priority Control Unit

extracts vehicle features from the time-history signatures and routes the information to the microprocessor via the Direct Memory Access (DMA) unit.

b) DMA - The DMA was developed under this project to provide direct access to the microprocessor memory from the preprocessor.

c) Microprocessor (μP) - The microprocessor is an off-the-shelf, 16-bit microcomputer unit marketed by National Semiconductor as the IMP-16 series. For this project, the capacity of the microcomputer included 4.5K of read-only memory and 4K of random-access memory, of which only 1K is used.

d) Programming Panel - The programming panel allows the traffic engineer to provide essential input to the microprocessor. This input enables selection of bus preempt/non-bus preempt, cycle length, main-street-green split, interval selection, and offset selection. These selections can be independently implemented for the three manual or loop-master selected timing plans.

e) Status Display and Monitor Unit (SDMU) - This unit interfaces the microprocessor bus classifier and traffic controller with the signal load relays. The SDMU also includes a watch-dog timer and essential interface to and from the conflict monitor.

The remainder of the components depicted in Figure 2 includes solid state load switches, conflict monitor, flash unit, and transfer relay. The conflict monitor, developed under contract to Honeywell and used with the Bus Priority Control Unit, provides "green" and "walk" conflict monitoring, minimum green and yellow interval monitoring, and maximum cycle length monitoring. If the Bus Priority Control Unit, or other peripheral equipment fails to meet the criteria of the conflict monitor, the conflict monitor will immediately transfer the intersection to a flashing operation. The flashing interval is timed by the transfer logic and then control is returned to the city controller at the dial position where transfer had been made to the BPCU. Recall that the city controller timing is stopped by a signal from the transfer logic when control is transferred to the BPCU by depression of the switch on the Transfer Control Panel. For manual transfer of the intersection from the BPCU to the city controller, the BACKUP switch is depressed on the Transfer Control Panel. The transfer logic then times a minimum main street green interval and control is transferred to the city controller with no intervening interval of flashing operation.

The entire system is housed in a "P" type cabinet and is depicted by Figure 9.

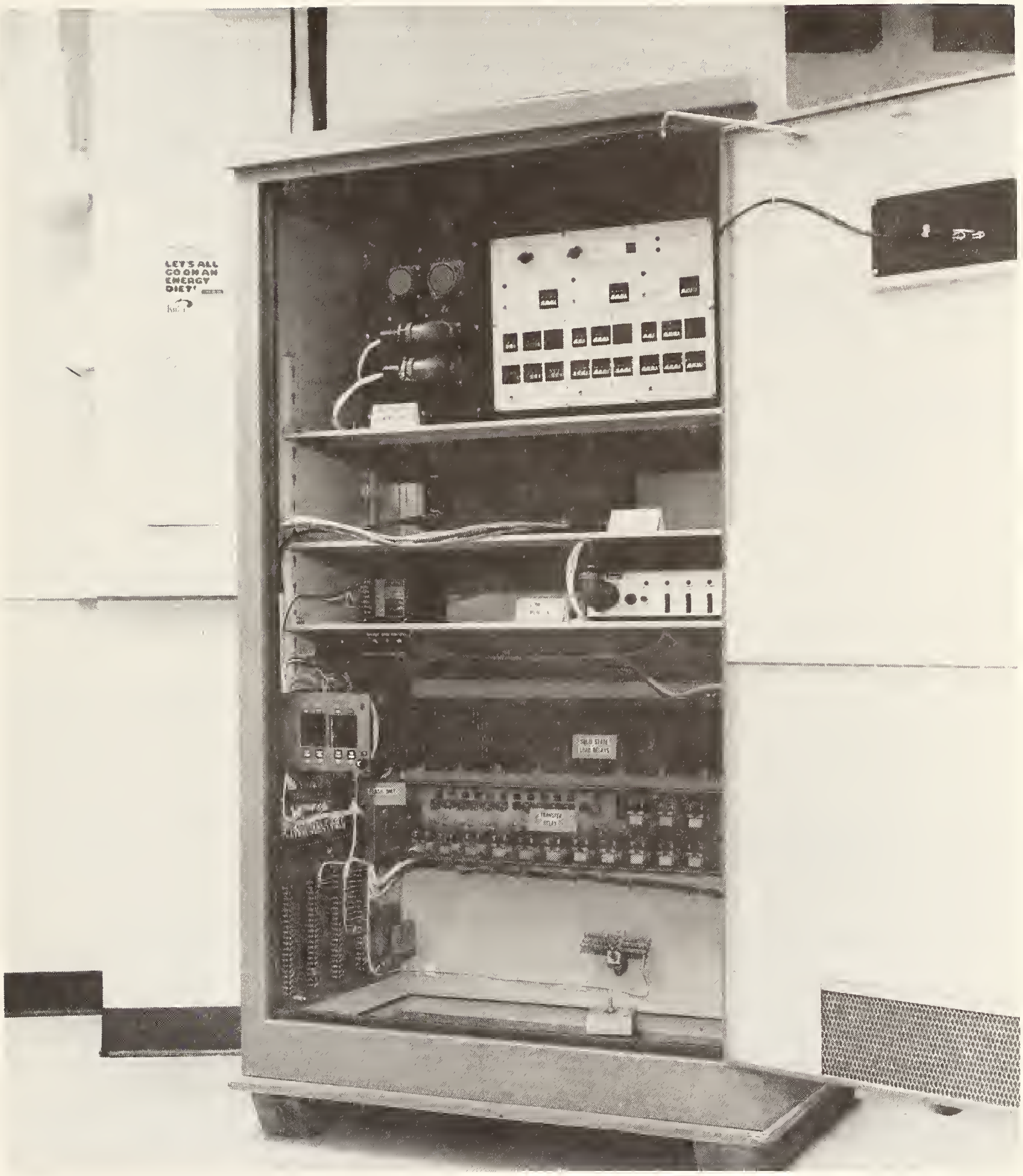


Figure 9. Passive Bus Detector/Intersection Priority System Engineering Unit

# SECTION III
# PROJECT OBJECTIVE

The objective of the Vehicle Detection Project, Phase III, was the development of a feasible Passive Bus Detector/Intersection Priority System hardware/software design and two engineering models. The engineering models were to be suitable for eventual installation for demonstration and evaluation at two typical urban intersections.

This objective was successfully attained during the course of the project.

# SECTION IV
# TECHNICAL DISCUSSION

This section provides a detailed discussion covering development of the Passive Bus Detector/Intersection Priority System. The discussion is concentrated in four parts:

a) Development of a Passive Bus Detector

b) Bus Priority Control Unit - Software

c) Bus Priority Control Unit - Hardware

d) System Conflict Monitor and Ancillary Equipment

Vehicle signature acquisition activities performed during the passive bus detector development are referred to Volume III of this report. Each of the above system parts is presented independently as follows:

a) Technical Objective(s)

b) Technical Approach

c) Progress and Results

## SYSTEM TECHNICAL OBJECTIVES

### Development of a Passive Bus Detector

The following are the technical objectives for the development of a passive bus detector:

a) Develop hardware and software that will identify an urban transit bus from all other urban traffic

b) Select transducer best able to meet the following bus detector design goals:

- 0-60 mph detection velocities
- 95 percent correct identification of busses
- 2 percent false-alarm rate
- Only standard 40-ft passenger transit busses manufactured by GMC, Flxible and AM General are to be considered as

busses. Performance for all other bus types (mini, articulated, intercity, tour, etc.) is not covered by the design goals.

- Bus classification when the bus centerline is within ±4 ft of the transducer centerline.
- 6 ft or greater separation between vehicles.

Bus Priority Control Unit - Software

The general technical objectives for the first-generation Passive Bus Detector/ Intersection Priority System software were:

a) Passive bus detection (standard 40-ft transit bus)

b) Traffic signal preemption

c) Pretimed traffic signal control (a minimum of three fixed-time plans)

d) Limited failure detection

e) Limited on-site signal plan programming

Specific requirements for the engineering model field test software are described in the following paragraphs.

Principal differences between the engineering model hardware and a prototype model are in the areas of environmental qualifications and stand-alone capability of the microprocessor system to control the signals. The engineering model is designed to operate with an existing controller as a backup to ensure reliable operation during field test operation.

The following are specific technical objectives which are addressed during the project:

a) Number of Detectors - Two passive bus detectors per lane will generally be required. Incertain applications, only one detector may be necessary. The design objective for the system was the capability to handle a minimum of eight lanes (16 detectors).

b) Bus Preemption - Two options for the bus preemption control strategy are incorporated. A "green time extension/advance" strategy is used for applications like the Minneapolis field test (similar to FHWA BPS algorithm, Reference 1). An "interval replacement" strategy is used for applications like the Washington D. C. field test in conjunction with a special bus interval. Either option is applicable to mixed traffic or exclusive bus lane operation. The option selected can be preprogrammed for a particular intersection. Figures 10 and 11 depict the street configuration for the two field tests.

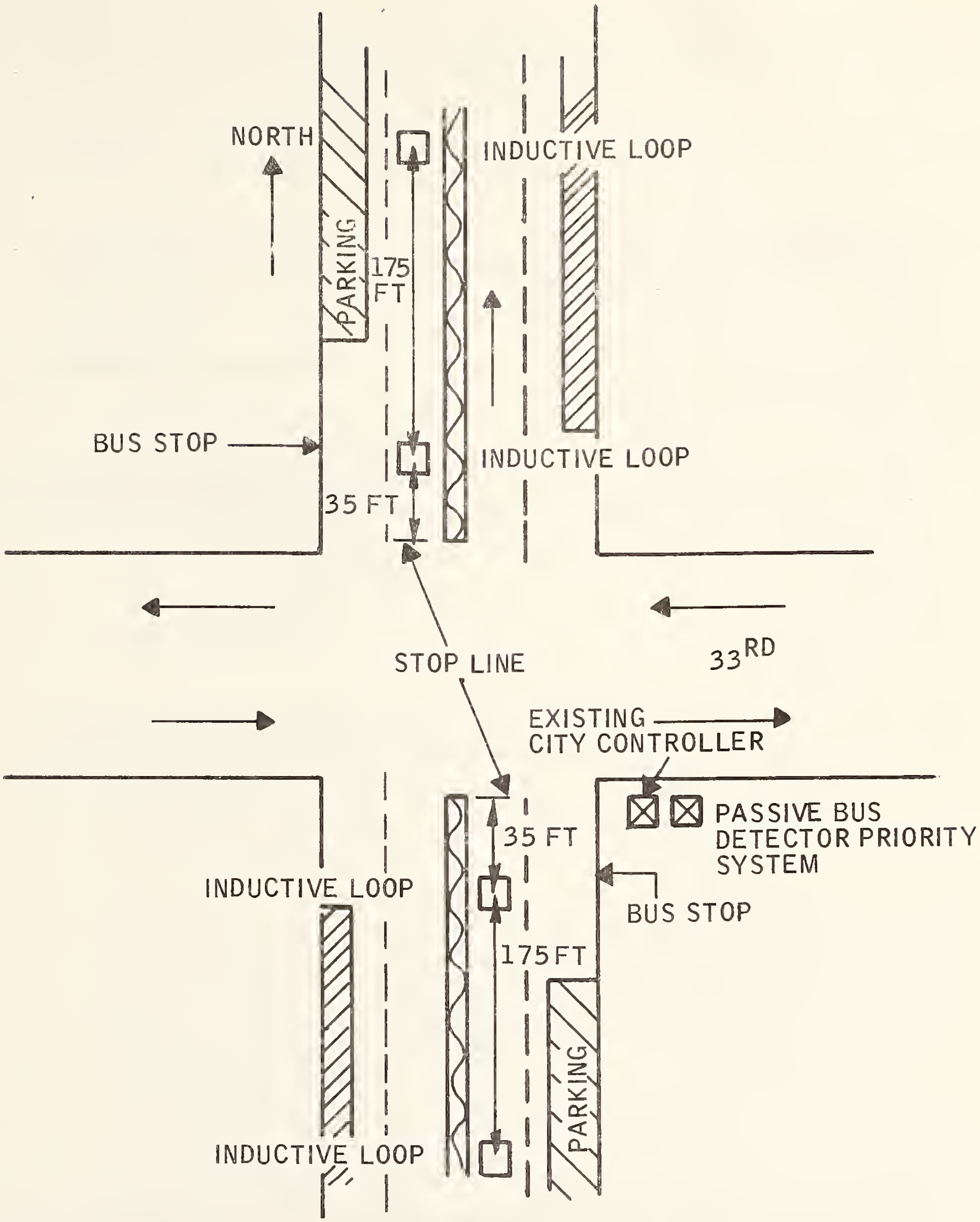


Figure 10. Installation and Demonstration, Minneapolis-33rd and Johnson St. NE

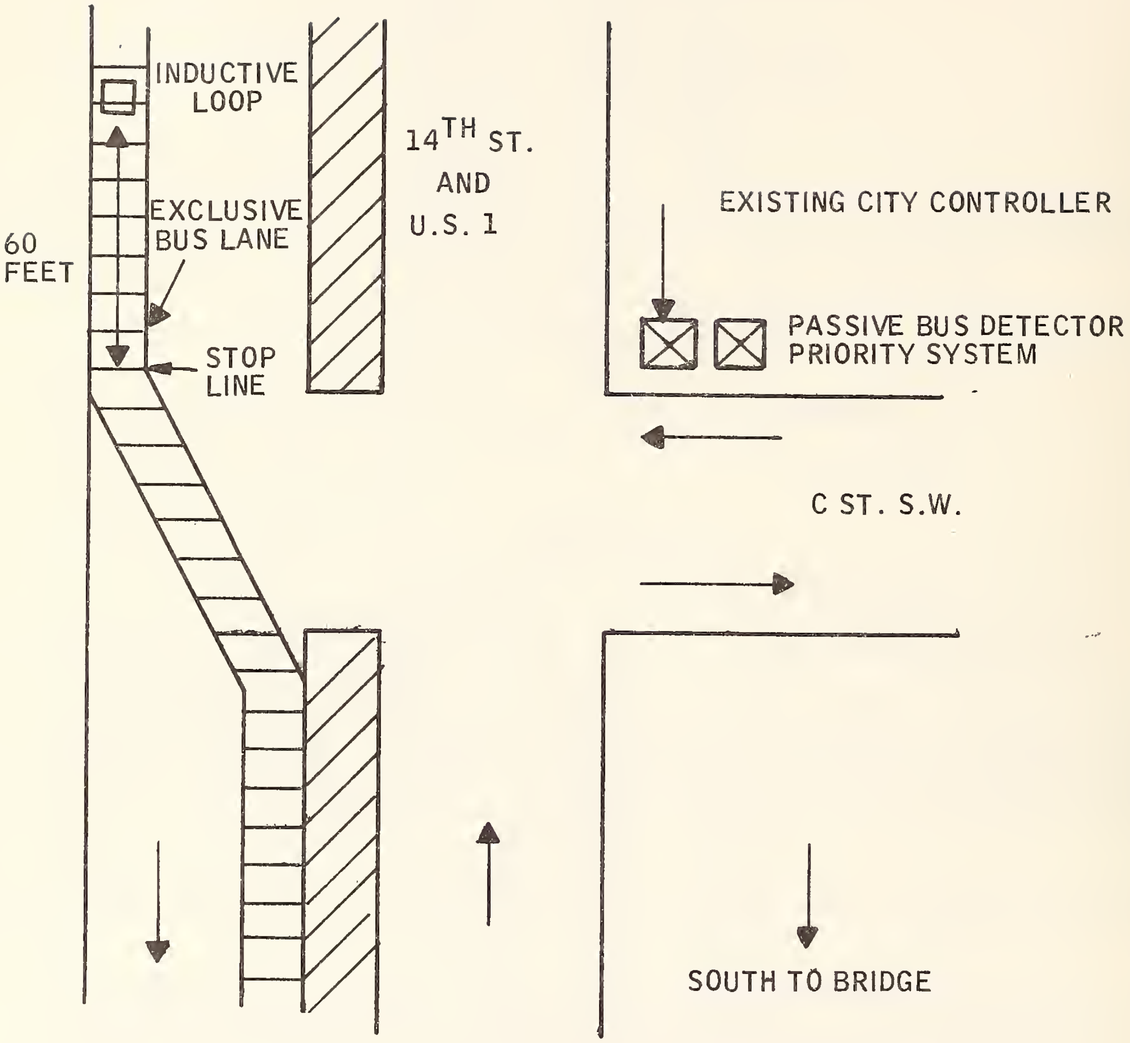


Figure 11. Installation and Demonstration, Washington D. C., 14th and C St. SW

c) Number of Traffic Signal Intervals - The software is designed to handle up to 16 timing intervals. The interval sequence is preprogrammed.

d) Number of Stored Timing Patterns - The software is designed for three cycle lengths and three offsets per cycle length.

e) Number of Circuits - The software and interface is designed to control up to 24 individual circuits.

f) Programming Panel - A limited on-site programming capability is provided through the programming panel (see Figure 44). For each of the stored timing plans, the following input is required:

- Cycle time
- Three offset times
- Main-Street-Green split
- Two special interval times related to the bus preempt
- Bus preempt on/off
- Synchronization on/off
- Manual override

g) Failure Monitoring - Five failure monitoring techniques were incorporated into the system design:

- Independent Conflict Monitor: A specially designed independent conflict monitor is incorporated to ensure safety.
- Microprocessor Self Check: The 4K RAM is tested continually in the background computation cycle during real-time control operation. Addresses not used for the data base or reserved for the catch-on-the-fly capability are tested. During initialization the 4K RAM and the RAM on the CPU card are tested prior to starting real-time operation.
- Watchdog Timer: The interface unit between the µP and the signal heads contains a timer to detect a microprocessor stall condition. If the microprocessor stalls, the BPCU is disconnected from the signals and control will revert to the Minneapolis or Washington, D. C. controller.
- Under Counting Test: Bus detector and preprocessor operation is continuously checked by means of an under-counting test. A vehicle count for each detector in the system is accumulated during a five minute interval and compared with a user-specified threshold value. An under-count indicates a problem with the detector or the preprocessor. An external status light is illuminated when an error occurs.

- Interface Test: A limited test of the microprocessor signal interface is performed by reading back into the microprocessor the contents of the output registers to the solid-state light drivers and comparing the input data with the output data.

If a failure is detected, (except in the case of the Under-Counting Test) control of the intersection is immediately transferred to the existing city controller. Manual restart is then required.

h) Power Interruption Response - The microprocessor power supply is designed to accommodate short-duration power interruption. The BPCU is designed for automatic restart after a power interruption.

i) Signal Coordination - The system is designed to operated in a coordinated manner (signals from a loop master) or as an isolated controller with manually selected signals plans.

## Bus Priority Control Unit - Hardware

The technical objectives for the BPCU hardware were as follows:

a) Develop microprocessor-based hardware which will perform bus classification and output traffic signal light control functions.

b) Develop interfaces to integrate the bus detector hardware and traffic light control (system conflict monitor and ancillary equipment).

c) Use hardware which is off-the-shelf.

d) Keep system cost to a minimum without sacrificing performance design goals.

## System Conflict Monitor and Ancillary Equipment

The objective of the system conflict monitor and ancillary equipment is to provide the BPCU capability to control signals at a typical urban intersection. To provide this capability, significant component technical objectives were satisfied, as follows:

a) Provide means of reliably switching 115 vac signals using 5-volt d-c logic from the BPCU.

b) Provide failsafe operation by using the Conflict Monitor Unit uniquely suited to the BPCU.

c) Implement acceptable response to a conflict.

d) Provide means to integrate the system at an intersection having an existing city controller.

e) Provide a suitable environmental enclosure which allows ease of component access and instrumentation hook-up, as necessary.

## TECHNICAL APPROACH

### Development of a Passive Bus Detector

The concept chosen for identifying urban transit busses was to develop a signal classification algorithm to operate on the signal output from a transducer using a buried sensing element.

The major steps in designing this algorithm were:

a) Record signatures (analog signal output of transducer) for statistical sample of busses and other urban vehicles (see Volume III).

b) Based on study of the signatures, select transducer with greatest potential for meeting design goals.

c) Using computer simulation:

- Determine criteria for detecting vehicle presence
- Define candidate signal features.
- Measure the feature values for each recorded vehicle signature.
- Determine equations for the bus/non-bus decision.
- Optimize chosen method for real-time implementation.

In addition to the major task of designing the classifier algorithm, the following analyses were performed:

a) Algorithm for estimating bus speed from the inductive loop detector phase output.

b) Analyze data from the inductive loop detector frequency output (FM) to determine its value for vehicle classification.

c) Analyze MGVD amplitude signatures to compare leading edge slope with vehicle speed.

## Bus Priority Control Unit - Software

The BPCU software is designed as a real time control program with the flexibility for the field test of the engineering concepts in passive bus detection and preempt operation.

Software functions are assigned to the basic computer program components (CPC) of Executive, Preempt, Timing Plan Execution, Data Input and Data Output. Within each CPC, the functions are contained within modular coding sequences.

The primary computing sequences are initialization, real-time control and background. As shown by Figure 12, the initialization sequence is entered at power up of the BPCU. Functions in this sequence start the background computations and start the real-time clock. The background computations are a self-test cycle which terminates only in the event of a failure condition. The real-time clock generates periodic interrupts for initiating the real-time control sequence. The 10 Hz clock interrupts provide for preempt modifications within a tenth of a second and timing interval measured to a tenth of a second.

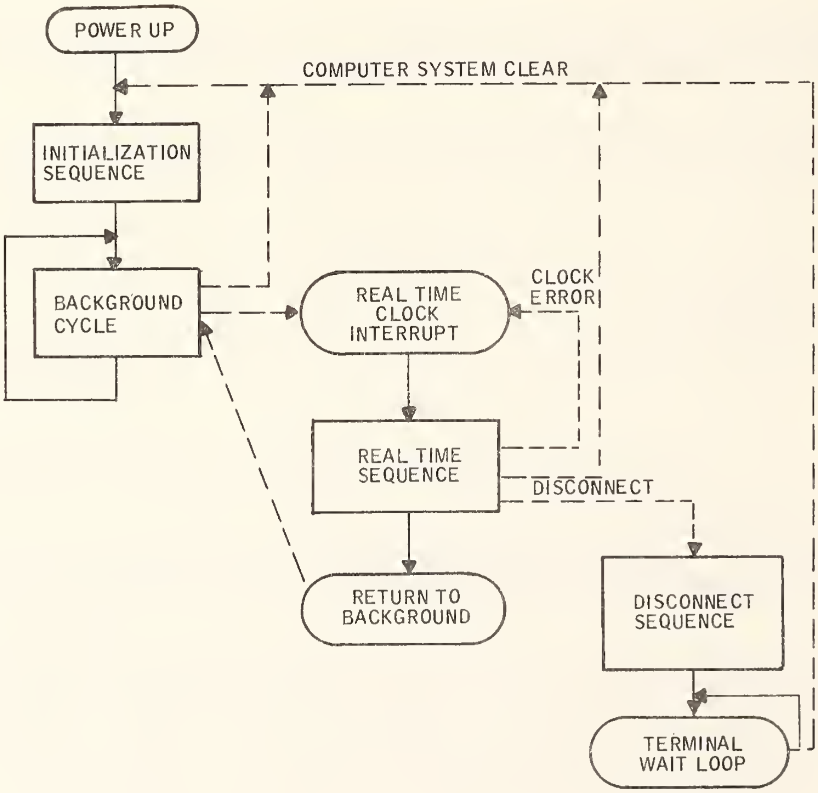


Figure 12. Software Operation

Integrity of the BPCU operation is monitored in several ways. Self-test operations in the Microcomputer Control Unit (MCU) software assure correct signal commands. Both the software and the hardware have autodisconnect capabilities. The existing controller is used as a backup and the MCU is programmed for the transfer of control operations. Internal error logging and external status display provide information on BPCU operation.

The MCU responds to manual inputs during real-time operation from a programming panel and the computer control unit (CCU). The programming panel provides on-site specification of timing plan, preempt and synchronization parameters. The CCU provides the computer control functions of displaying memory and register contents and altering read/write memory contents.

The specification of timing plans in the MCU data base is designed for modification by the preempt functions. Fixed intervals, which must always be timed, provide for safe operation of the intersection and clearance of all traffic flows. Extendible and contractible intervals are added to provide the modifiability for preempt extension of a green sequence and pay back of unused extension time. Timing plan execution allows for zero-duration intervals which may appear anywhere within the timing plan. The preempt operation is generalized such that data parameters relate the detector configuration to the modifiability of the timing plan intervals. The contractible intervals and the extendible interval are specified as data for each detectorized approach to the intersection.

The vehicle classification algorithm is designed for reliable and efficient operation. Each signature is tested in several ways for validity and non-bus signatures may be rejected at the front end of the algorithm. The signature is tested for a minimum duration. Signatures whose presence time is too short are rejected as noise. Signatures with too few features are rejected as non-busses. Data in the signature are tested for being within the proper range and time features are tested for being within the proper sequence. A portion of the detailed classifier handles signatures of vehicles having uniform speed within the time interval data range. Another portion of the classifier handles signatures of busses which accelerated, decelerated or stopped over the detector.

## Bus Priority Control Unit - Hardware

The technical approach for the BPCU hardware is summarized by the following:

a) Microprocessor classifier and signal light control

- Partition classifier and signal light control functions.
- Assign functional requirements to each partition.

- Review available technology.
- Find specific hardware to meet the functional requirements of both microprocessor functions.

b) Interfaces for bus detectors and traffic light control

- Determine the specific kinds of interfaces along functional requirements.
- Determine individual interface specifications.

System Conflict Monitor and Ancillary Equipment

The technical approach adopted for the developme nt of these components can be divided between the conflict monitor and ancillary equipment. The system conflict monitor was under contract to Honeywell and used in this project.

The ancillary equipment designed approach concentrated on using established principles used in traffic controller and controller cabinet designs to ensure compatibility with established procedures and familiarity to the traffic engineer.

## PROGRESS AND RESULTS

### Development of a Passive Bus Detector

Summary

The classifier algorithm was designed by analyzing the characteristics of recorded vehicle signatures. These signatures were produced by test vehicles driving over experimental installations in Minneapolis and in Washington, D. C. and by actual street traffic in Washington, D. C. A number of experimental parameters that could affect the vehicle signatures were varied during the experimental runs. These include vehicle parameters (speed, centerline displacement, separation, etc. ) and installation parameters (asphalt, concrete, reinforcing mesh, lead wire length, snow pack, etc. ). Also, signatures were recorded for vehicles that stopped over the sensing element. Data acquired at the Honeywell vehicle test facility are detailed in Volume III.

The initial data recorded was the amplitude and the phase of the signals from both the Magnetic Gradient Vehicle Detector (MGVD) and the Inductive Loop Detector (ILD). Recordings of magnetometer signatures were available from a previous program (see FHWA-RD-7522). This data was studied to determine which sensing method was most appropriate for vehicle classification. Only the MGVD amplitude and the ILD phase produced signals that could be easily identified by eye as being bus or non-bus. For vehicle runs near centerline, these two transducers produce similar-appearing signatures, but for displacements of three to six feet, the MGVD amplitude signals had both positive and negative values. Since the two methods appeared to have the same classification potential with the ILD phase showing fewer problems for classifier development, the loop phase transducer was selected.

A number of signal characteristics were apparent in the initial recordings and these were used to make a preliminary definition of the classifier structure. First, all vehicles produced only positive signal values with peak values related to vehicle mass and inversely related to road clearance. Thus, a signal threshold would detect vehicle presence and would reject most bicycles and motorcycles. Second, vehicles with short wheel bases produce a signal that has only one peak value, while long-wheel-base vehicles show multiple peaks. This suggested that if the multiple peaks were consistent and could be measured, then a multi-stage classifier structure could reduce the real-time data processing rate by counting peaks and rejecting most automobiles. The vehicles that pass this gross test are then classified by using additional, more detailed features. This classifier structure is shown in Figure 13.

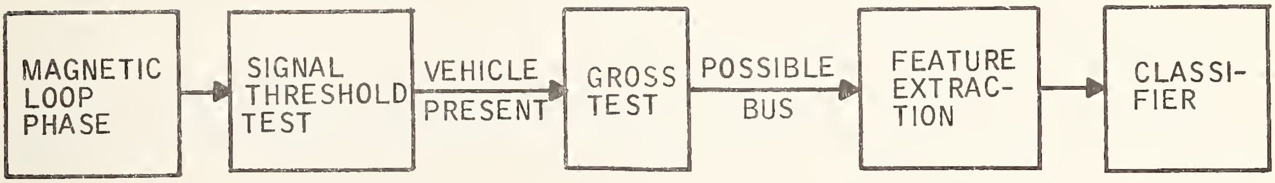


Figure 13. Multi-Stage Classifier Concept

At this stage in the development, it was known that there would be an analog signal in the buried ILD and that the final classification would be performed digitally, but it was not known where the change from analog to digital would occur. A similar uncertainty thus plagued the design process. Ideally, the design would be conducted by computer simulation of the signal and data processing functions. The best simulation of an analog function is done on an analog computer. The design of the final classifier stage has to be done digitally, but the other functions in Figure 13 can be simulated by either analog or digital methods. These considerations led to selecting full hybrid computer (PACER 700) for the simulation. The philosophy chosen was to input the analog signals, digitize them and perform the total simulation digitally. As each stage was designed, it could be tested digitally; and if it then appeared advantageous to perform that stage by analog means, the analog functions could be set up on the same computer and the analog-to-digital conversion moved to the output of that stage.

The final design was all digital but in retrospect, the hybrid approach provided other advantages. It was possible to process analog recordings at up to 16 times real speed. Thus, a 3-hour recording could be analyzed in 12 minutes. Being able to input analog recordings was a great timesaver in the final parameter tradeoff analysis and in designing the digital preprocessor. Finally, the fact that the analog input was not synchronized to the digital processing revealed problems that would not have appeared in a simulation using predigitized signals. These were then solved in the simulation phase rather than being discovered in the first breadboard model.

The design procedure started by simulating the signal threshold test on the PACER. Having determined the threshold value, the next step was to define an algorithm for detecting signal peaks. A number of algorithms were formulated and tested on the PACER. The one chosen was able to detect all the major peaks that were obvious to the eye (independent of the shape of the peak) and it detected a minimum number of spurious peaks (small amplitude signal variations and noise effects). The gross test was simulated next. Eyeball study of signal tracings suggested that busses within four feet of center line produced from three to six major peak values. This test was simulated and a number of tests were run against recorded data using various lower and upper limits to verify this criterion.

Up to this point all the work had been done using the hybrid simulation and manually analyzing the results of tests performed with recorded signals. Preliminary values had been established for the vehicle detection, threshold, the peak detection threshold and the upper and lower limits for the gross test. These three processing steps passed all of the bus signals and eliminated more than 90 percent of the non-bus events that would be encountered in normal traffic. What remained to be done was to find appropriate signal measurements (features) that would let us distinguish between busses and the other vehicles that passed the gross test. Moving vans, semis, and non-urban transit busses are examples of the vehicles to be eliminated. The signals from these other vehicles showed roughly the same signal shape as the busses. There was also the problem of signal shapes being distorted by a vehicle changing speed or stopping.

By plotting peak values versus time for a number of different test runs it was seen that these signal characteristics contained enough information for a human to discriminate between various vehicles moving with uniform speed. It was decided to attempt a classifier design using the peak values as features. There had also been an effort, prior to that time, to find a classifier structure that would group together all functions that differed only by a continuous monotone transformation of the independent variable. For vehicles moving along the center line of the loop, the detector output is a function only of the vehicle distance x along its path. Denote this by f(x). The distance x is, in turn, the integral of the velocity v(t) which is an arbitrary continuous function of t, thus

$$x(t) = \int_{t_o}^{t} v(t)dt$$

Since v(t) is assumed to be positive (i. e., the vehicles are always traveling in one direction), x(t) is therefore an arbitrary positive valed, continuously differentiable, non-decreasing function of t. If the same vehicle crosses the loop twice with two different velocity functions $v_1(t)$ and $v_2(t)$, the output of the detector will be two different time varying functions that take on the same values but at different times. Since we are interested in the output independent of speed variations, we would like to deal with only those aspects of the function f(x) that can be expressed and measured independent of the actual velocity. Classifier structures invariant under continuous monotone transformations are an optimal set of mathematical structures in which to treat this problem.

This would have allowed both moving and stopping vehicles to be classified by one classifier. The effort revealed that such a classifier would reduce the number of independent time measurements to just one, for non-linear transformations (non-uniform speed). This then led to a decision to use two classifiers, one for uniformly moving vehicles and one for stopping (or non-uniformly moving) vehicles. The time measurements then served to determine which type of motion was being observed and, for uniform speed vehicles, the time measurements would provide a speed estimate and thus allow the effect of speed differences to be removed. The final uncontrolled non-significant variable was the change in signal magnitude caused by the displacement of the vehicle from the loop's center line. It was decided that this effect could be treated as a signal attenuation factor that did not change the signal shape and its effect was removable by normalizing the signal amplitudes.

The amplitudes of the signal peaks normalized by dividing each peak amplitude by the amplitude of the first peak. Thus, amplitude of the first peak of the normalized signal is always equal to 1. The measurements of peak magnitudes and times (primary features) thus had to be transformed or normalized into values called secondary features that were independent of vehicle speed and lateral displacement. The normalization is described in the following paragraph.

To perform the feature analysis and the classifier design, the primary features generated by the PACER hybrid simulation were punched on cards for input to the XDS9300 digital computer. A number of methods for normalizing the peak magnitudes and times were tested using the primary features from vehicles that passed the gross test. It was found that the first signal peak was consistently measured by the hybrid simulation and the ratio of Nth peak magnitude to 1st peak magnitude provided the necessary magnitude normalization.

Normalizing the time values was not as easy. A velocity estimate was the ideal normalizing factor, but this required knowing the length of the vehicle and that, in turn, required knowing what kind of vehicle made the signal. An attempt was made to use the total signal duration, but this caused all vehicles to have the same signal length and produce confusion between long and short vehicles. The normalizing factor finally used was the sum of the time from vehicle detection to first peak plus the time from last peak to end of detection. The primary time features were divided by this factor to produce the (normalized) secondary time features. The normalized time from detection to first peak was found to be near 0.5 for all vehicles moving with uniform velocity, thus the moving versus stopped vehicle test was based on this value. (The test was later modified to separate uniform and non-uniform speeds. The non-uniform speed classifier than was concerned with accelerating and decelerating vehicles as well sa stopping vehicles.)

One more decision was made to simplify the classifier structure - that only the first, second and last peaks (magnitude and time) would be used to generate secondary features. Any peaks detected between the second and the last would be ignored.* Since every signal passing the gross test showed three or more peaks and since all the recorded bus signatures had three major peaks, it was felt that this constraint would not limit the classifier performance. In terms of classifier structure, each primary feature set contained the same number of measurements, regardless of how many peaks were detected. The classifier equations, therefore, did not have to account for the varying number of peaks. The validity of this choice and the normalizations were established by the classifier performance.

The secondary feature set then had two normalized magnitudes and two normalized time intervals. A graphical/statistical analysis was performed to determine whether these secondary features contained enough information to distinguish between busses and other vehicles. For this analysis, the thresholds in the hybrid simulation were set to allow an abnormally large number of non-bus runs to pass the gross test. This provided signatures from the three busses that had been used, plus signatures from six other vehicles (Volkswagen, station wagon, step van, dray truck, semi and moving van). Each of these was treated as a separate vehicle class and the sizes and locations of these nine class cluaters was calculated. The results were that eight clusters were located close to a two-dimensional plane in the four-dimensional space of secondary features (the moving van was the maverick),

*Seven memory locations are used to store the three peak magnitudes and the four time values. Denote these locations by ($M_1$), ($M_2$), ($M_3$), ($T_1$), ($T_2$), ($T_3$), ($T_4$). When the first peak is detected, its values are stored in ($M_1$), (T ). The second peak is stored in (M ), (T ). If there are more than two peaks then they are stored in (M3), (T3) with each new peak writing over any information previously stored there. The time at the end of the signal is stored in (T4). Thus, the last detected peak appears in (M3), (T3) at the end of the signal.

the three bus clusters and the moving van cluster were close to each other but far from the other five clusters.

With this evidence in favor of the four secondary features being sufficient, the next step was to determine equations for a classifier. The method used was to arbitrarily assign a number to each cluster of data samples (for example: 1 denotes bus, 2 denotes semi, 3 denotes truck, etc.). These are called "class values". Then, considering the class values as a function defined at the sampled points in the four-dimensional space of secondary features, a linear function (called a "Discriminant") is found that gives the best least-squares fit to the class values. (Ideally this function will have near 1 for all the points in cluster number 1, near 2 in cluster 2, etc.) For the general multiclass problem (more than two classes) this method requires that the class values be permuted to determine the minimum least-square error, but our problem involved only two classes (but and not bus) so this step was not necessary. Having determined the discriminant, the classification decision was made by comparing the function value with a fixed threshold value; small values give one decision, and large values the other. For the bus classifier it was not possible to find a single discriminant that would correctly classify all of the experimental data. In the two best cases, one of them classified some of the moving van examples as bus and the other called all of the trucks busses. However, the combination decision rule, that decides "bus" when both discriminants indicate "bus", correctly classified all of the experimental data. Figure 22 shows the two-dimensional distribution of the test data. The region to the left of 2.89 and below 1.57 is the bus region. Neither threshold by itself can separate the bus samples from the others.

The preceding discussion refers to the moving vehicle classifier. For the stopped vehicles, only the normalized signal magnitude features were used. For a stopped vehicle, one of the time intervals becomes arbitrarily large and thus does not characterize the vehicle class. The method for finding discriminants was the same as described above, but the decision rule was slightly different. In this two-dimensional space the clusters all lay near one line with the nonbusses at the outside and the busses in the middle. Thus an upper and a lower threshold on the discriminant values was needed. Again, two discriminants were combined to correctly classify all of the experimental data. Figure 23 shows how the moving vehicle discriminant values and the thresholds are arranged.

The remaining problem for the classifier was to decide whether a detected vehicle stopped or moved continuously over the loop. For a moving vehicle the normalized time $E_1$ from detection to the first peak was near 0.5 Thus a test of the form $0.5 - \alpha \leq E_1 \leq 0.5 + \beta$ indicates that the vehicle probably did not stop over the loop. Unfortunately this condition while necessary, was not sufficient to distinguish between them. An additional value $E_2$, the normalized time from first peak to last peak, will be large if the vehicle stops while covering the loop. When the test $E_2 \leq \gamma$ was added as a condition for moving vehicles the results were satisfactory. Values for $\alpha$, $\beta$, $\gamma$ were determined from the distribution of $E_1$ and $E_2$ for moving and stopping vehicle experiments.

It was found that certain moving vehicles were more easily classified by the stopped vehicle classifier. Study of the signal recordings and logs revealed that these vehicles had been accelerating or decelerating as they crossed the loop. The values $\alpha$, $\beta$, $\gamma$ were then adjusted and the two final vehicle divisions are more appropriately called uniformly and non-uniformly moving vehicles.

Subsequent to the initial classifier design, additional experimental data were obtained from the FHWA facility in Washington, D.C., plus recordings of street traffic in Washington, D.C.. This data, plus the original data, were all processed through the hybrid simulation incorporating the full classifier and, after minor changes to threshold values, the classifier correctly identified all signals with the exception of three questionable recordings. The exceptions are discussed in more detail in the technical discussion along with other possible improvements to the current classifier algorithm.

## Bus Classifier

The bus classifier is neither a device nor a computer program, but rather a collection of functions operating together and being performed in a number of different locations in the bus detector. Figure 14 is a functional diagram of the bus classifier as it was simulated on the PACER 700 hybrid computer and as it was implemented in the preprocessor/microprocessor configuration. This figure shows in greater detail the same functions described in Figure 11 (Multi-stage Classifier Concept).

In its final form, the "sensor, amplifier and filter" functions are performed by the phase loop electronics. This, and the analog-to-digital conversion (A/D) are described in later sections of this report. The present discussion will consider the bus classifier to begin at the digital output of the A/D.

At this point, a time-varying digital value represents the signal output of the loop detector. In the absence of any vehicles passing over the loop, it is seen that the loop output varies slowly due to changes in the circuit Q caused by component aging or temperature changes. The value of this slowly changing bias is estimated by averaging the signal over long time periods when no vehicles are present. The averaging is done by infrequently comparing the value of a counter with the current signal value and incrementing (decrementing) the counter by 1 if the signal value is larger (smaller) than the counter value. The time constant of the resulting low-pass filter is thus determined by the time elapsed between comparisons. The effect is the bias is removed before any signal tests or measurements are made.

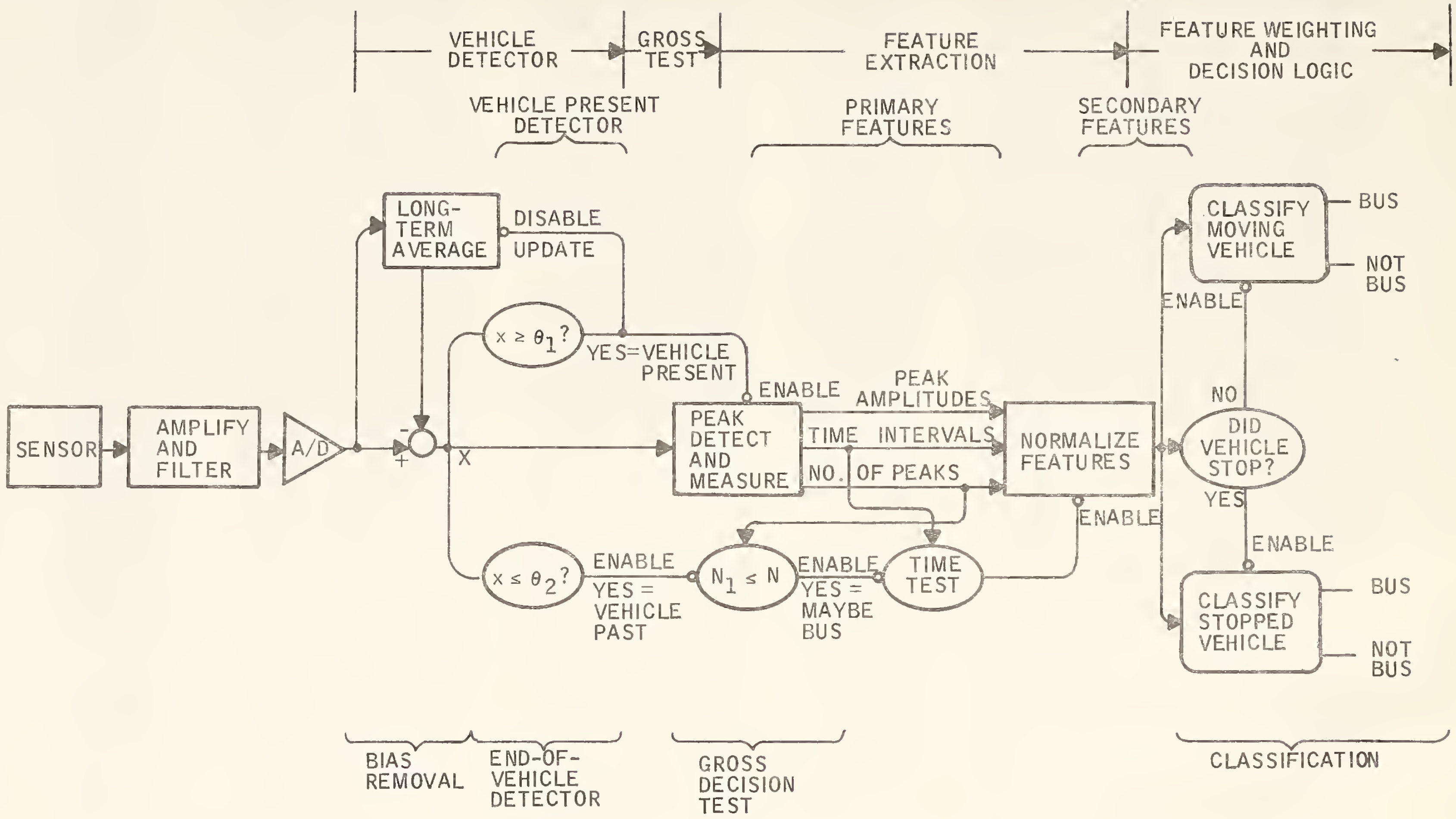


Figure 14. Functional Diagram of the Bus Detector

The debiased signal thus has a zero value when there is no vehicle present. When a vehicle is present, the biased signal becomes positive, reflecting the change in phase caused by the vehicle. Vehicle presence is thus detected by comparing x with a small threshold value $\theta_1 > 0$. A second threshold value $\theta_2 < \theta_1$ is used to determine when the vehicle is no longer present. Thus, the signal associated with one vehicle is all those values starting at the sampling time $t_o$ where $x(t_o) \geq \theta_1$ and continuing until time $t_n$ where $x(t_n) \leq \theta_2 < \theta_1$. (The condition $\theta_2 < \theta_1$ is used to prevent a single value from satisfying both conditions.) While a vehicle is present, the signal is observed and the primary feature values (peak times and magnitudes) are measured and recorded. At the end of the vehicle, the gross tests are performed to decide whether the signal should be classified or the signal was clearly not a bus. The gross test simply checks whether there were three or more peaks detected. If so, then the vehicle may be a bus and further study of the signal will be made.

The bus classifier functions described to this point are all implemented as software routines in the preprocessor. The primary features are measured by the preprocessor but their values are stored in dedicated locations in the microprocessor's memory. The following functions are performed in the microprocessor.

Following passing of the gross test, the primary feature values are normalized, yielding secondary feature values that are independent of vehicle speed and lateral displacement. The secondary time features are tested to decide whether the stopped vehicle or the moving vehicle decision algorithm should be used for classification. (The stopped vehicle algorithm uses only the normalized peak magnitudes, the moving vehicle algorithm uses both magnitudes and times.) The appropriate algorithm then decides whether the vehicle was an urban transit bus or something else. Detailed descriptions of these functions and their development follow.

Figure 15 shows some typical signal traces from the test vehicles used in designing the bus classifier. Outstanding characteristics of these signals are that, except for rare electromagnetic interference, there is no noise and the signals are almost exactly reproducible, depending only upon the position and location of the vehicle relative to the loop. (Whether the lack of noise will apply to all environments is unknown at this time. In the data recorded in Washington, D.C. at the testing location, their were many noise spikes apparently caused by the brake system on the bus. These were successfully ignored by the classifier by eliminating the maximum number of peaks test.)

For the purpose of distinguishing between the two busses and the other vehicles, there are some signal features that are obvious to the eye. The signals from the short vehicles (VW, Torino, station wagon) show a single "hump". The step van and the dray truck show two humps and the busses, moving van and

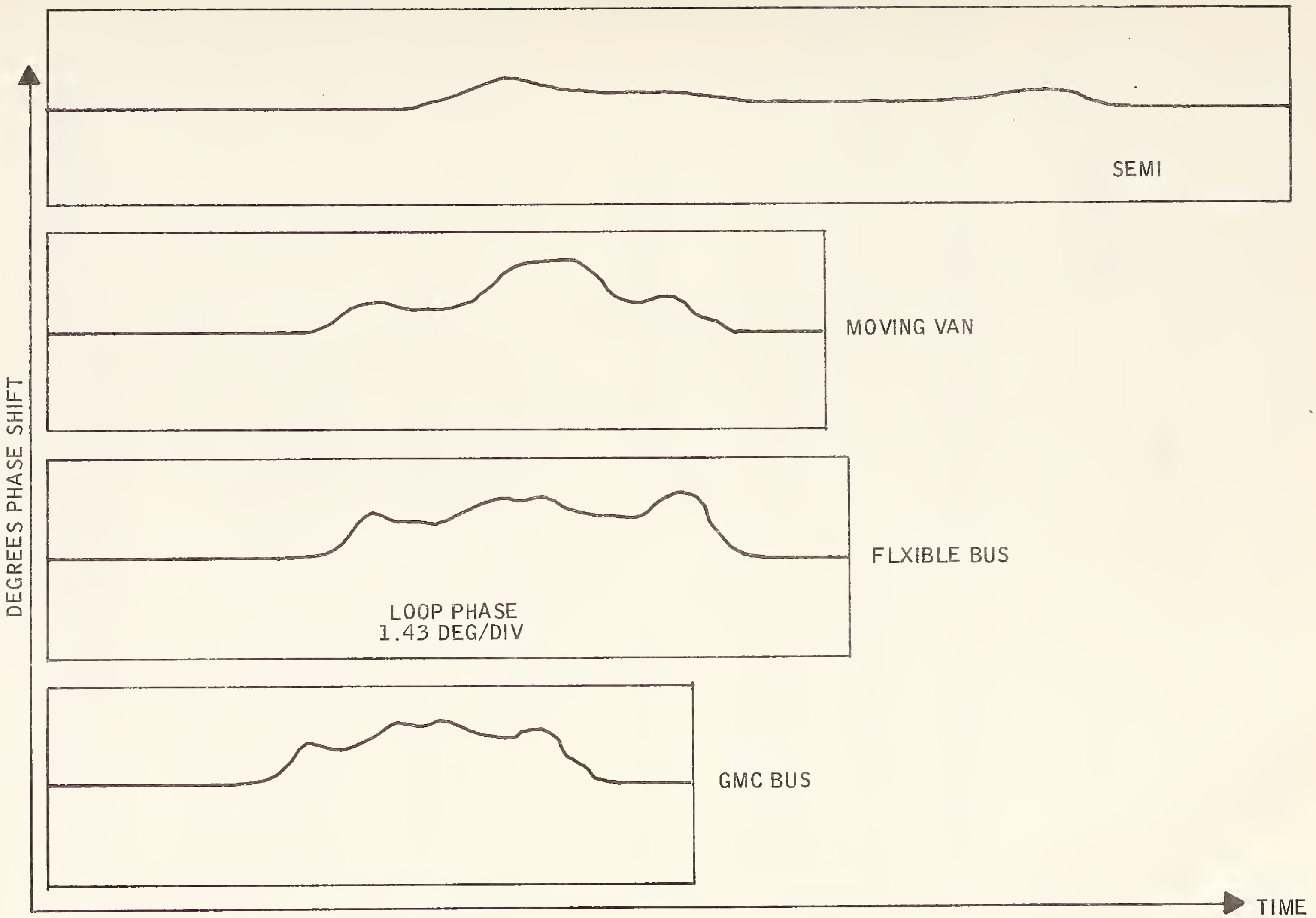


Figure 15. Vehicle Signature Examples

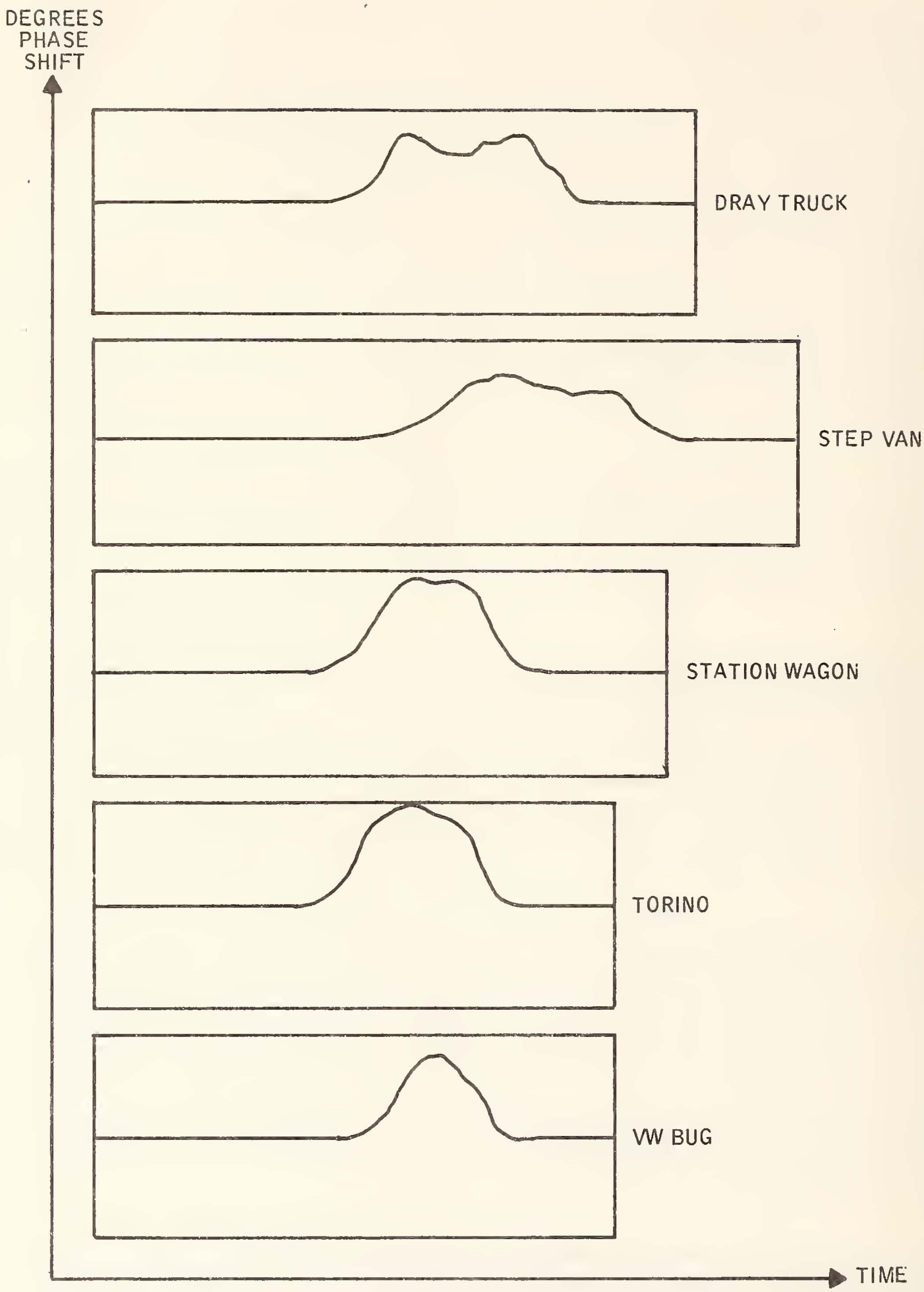


Figure 15. Vehicle Signature Examples (Concluded)

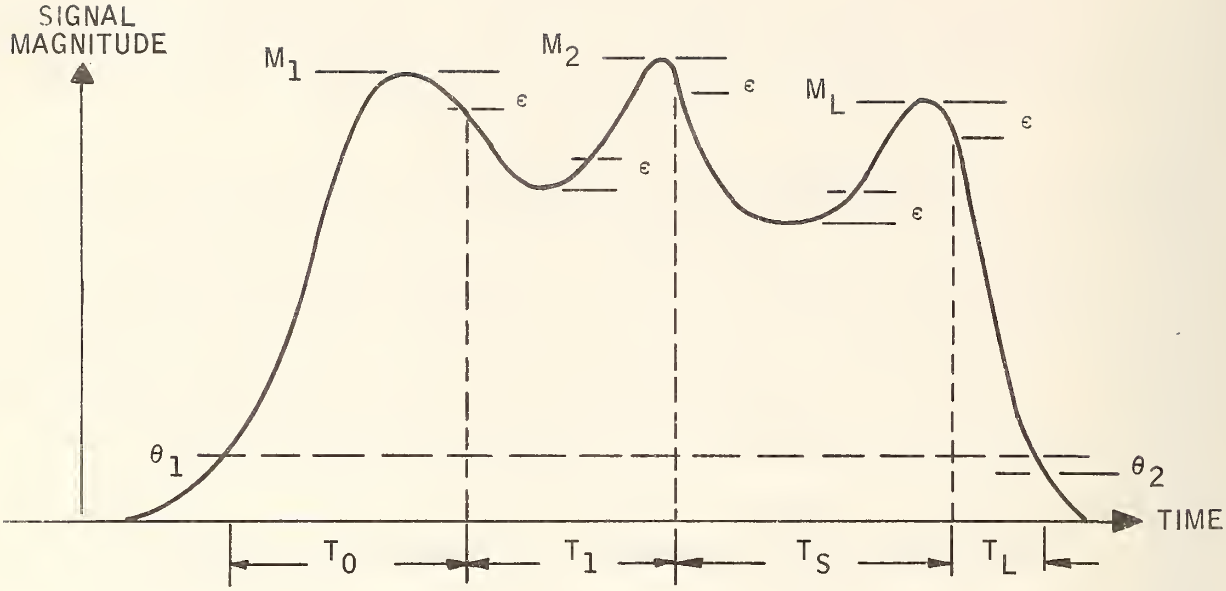
semi have three humps. (One might postulate that those humps are caused by metallic structures in the vehicle undercarriage that are large enough and separated enough to be resolved by the 6-foot by 6-foot loop.) The main classifier problem then seems to be in distinguishing between busses and other long-wheel-base vehicles. Looking again at the examples, the bus humps look evenly spaced and about of equal height while the moving van and semi are quite irregular. A first impression is that the signals can be classified by using the height and spacing of the humps. The problems appear in attempting to define "hump", "height", "spacing", "regular", etc. in a way that a machine can measure and use. The bus signals have only three main humps, but they have many small bumps and so forth.

Figure 16 describes the method used to measure the signals in a way that parallels the eyeball features described above. The thresholds $\theta_1$ and $\theta_2$ are as discussed earlier. The value $\epsilon$ is called the "peak tolerance" or simply the "tolerance" and is critical to detecting local signal maxima (peaks) that correspond to the "eyeball humps". To describe how a peak is detected, let the signal be sampled at times $t_n = n\Delta t$ where $x(t_o) \geq \theta_1$ is the first sample after detection. The peak detection algorithm starts with the peak magnitude estimate $M_{t_o} = \theta_1$. At each sample time $t_n$, the peak magnitude is updated according to Equation (1).

$$\begin{aligned} &\text{If} \quad x(t_n) > M_{t_{n-1}} \quad \text{then } M_{t_n} = x(t_n). \\ &\text{If} \quad M_{t_{n-1}} - \epsilon < x(t_n) \leq M_{t_{n-1}} \quad \text{then } M_{t_n} = M_{t_{n-1}}. \end{aligned} \qquad (1)$$

This updating is continued until the sampled signal value becomes less than the maximum minus the tolerance (i.e., $x(t) \leq M - \epsilon$). The time at which this happens is the time of the first peak and the signal maximum value up to that time is the magnitude. Following the detection of a peak, the algorithm searches for a local minimum using the same tolerance value $\epsilon$. Upon finding a minimum, the search for the next peak begins in the same manner as before. This algorithm thus ignores signal variations smaller than the tolerance $\epsilon$ and can therefore be made to detect only the major bumps that are obvious to the eye. The time measured by this algorithm is slightly later than the actual time of the maximum. This timing method was chosen for convenience in the simulation and an equally good classifier can be developed using the actual times of the maxima.

Initially, the magnitudes and times of all peaks were measured. Experiments with the data showed that the decision between bus and non-bus could be made using only the first, second and last peaks. This edited primary feature set is defined in Figure 16.



PRIMARY FEATURES

$T_0$ = TIME FROM TURN ON TO FIRST PEAK

$T_1$ = TIME FROM FIRST PEAK TO SECOND PEAK

$T_S$ = TIME FROM SECOND PEAK TO LAST PEAK

$T_L$ = TIME FROM LAST PEAK TO TURN OFF

$M_1$ = MAGNITUDE OF FIRST PEAK

$M_2$ = MAGNITUDE OF SECOND PEAK

$M_L$ = MAGNITUDE OF LAST PEAK

Figure 16. Primary Feature Measurements

The primary feature values are absolute time and voltage measurements and thus depend upon circuit parameters (gain, length of leads, etc.) and upon the speed and lateral position of the vehicle. These cause the time and magnitude scales to be linearly distorted and thus change the shape of the signals. There are also those vehicles that come to a stop or change speed while over the loop. These cause a non-linear change in the time scale and produce an even greater distortion of the signal's shape. Initially, an attempt was made to formulate a decision algorithm that could accommodate all these variables. This proved to be impractical and a second approach was tried. To take care of the non-linear distortion, it was decided to use a separate decision algorithm for the stopping or accelerating vehicles. This algorithm would use only the signal magnitudes and thus avoid the non-linearities. To handle the linear distortions, it was decided that normalizing factors should be used that would remove the gain and time scale dependencies.

Having made these decisions, the next step was to find ways to carry them out. Numerous experiments with features obtained from the test vehicle recordings lead to the following secondary feature definitions:

$$F_1 = T_1/(T_o + T_L)$$

$$F_2 = T_S/(T_o + T_L)$$

$$G_1 = M_2/M_1$$

$$G_2 = M_L/M_1$$

$$E_1 = T_o/(T_o + T_L)$$

$$E_2 = F_1 + F_2$$

$F_1$ and $F_2$ are the normalized values of the time from first peak to second peak and the time from second peak to last peak, respectively. $G_1$ and $G_2$ are the normalized magnitudes for the second peak and the last peak. Figure 17 shows the normalized signal averages for nine different test vehicles. These plots were made from runs that passed the gross test (three or more peaks) and they show the normalized magnitudes for the first, second and last peaks plotted at their normalized times. Comparison with the typical signals shown in Figure 17 reveals obvious correlations in the shapes, thus indicating that this normalization preserves a large portion of the information in the original signal. The last two secondary features, $E_1$ and $E_2$, are used to decide between stopping and moving vehicles. These will be discussed later.

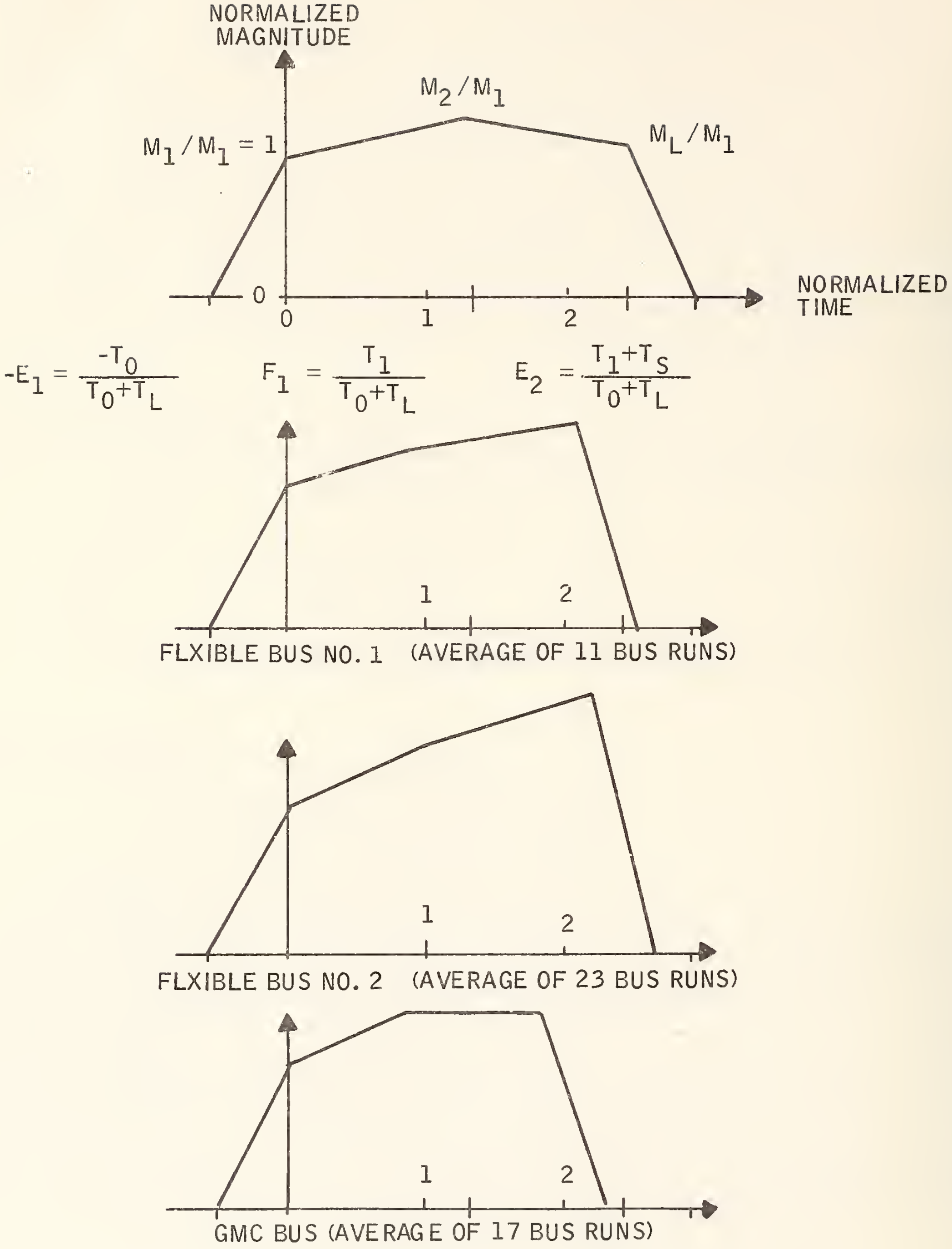
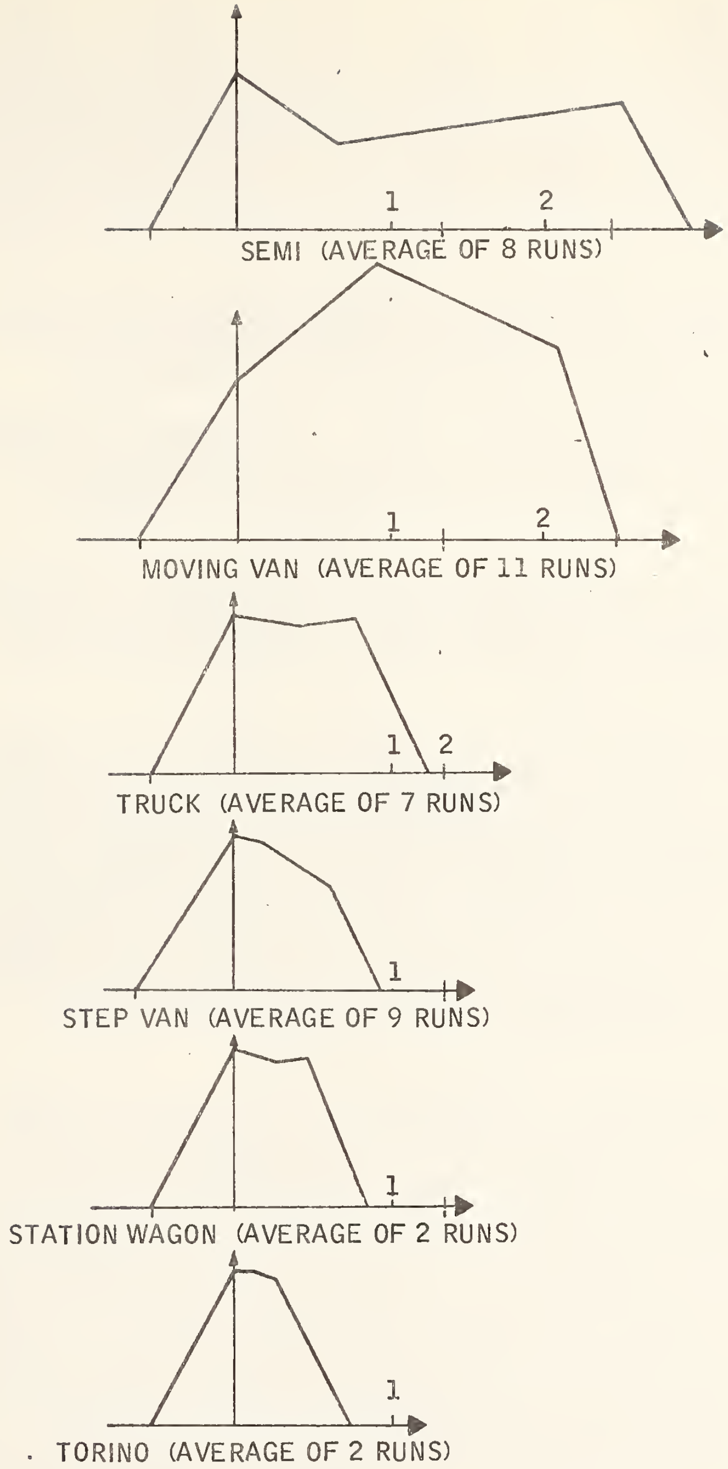


Figure 17. Normalized Signature Average



SEMI (AVERAGE OF 8 RUNS)

MOVING VAN (AVERAGE OF 11 RUNS)

TRUCK (AVERAGE OF 7 RUNS)

STEP VAN (AVERAGE OF 9 RUNS)

STATION WAGON (AVERAGE OF 2 RUNS)

TORINO (AVERAGE OF 2 RUNS)

Figure 17. Normalized Signature Average (Concluded)

An introduction to the basic pattern recognition concepts and terminology used in the following discussion can be obtained from many standard references in this area. The chapter references given below are closely related to the particular linear methods used in developing the bias classifier. Any one of these will provide sufficient background for the reader acquainted with elementary statistics.

- Sebestyen, George S., "Decision-Making Processes in Pattern Recognition", Chapters 2 and 5, ACM Monograph Series, Macmillan 1962 (Library of Congress number 62-19428).

- Arkadev, A.G. and Braverman, E.M., "Computers and Pattern Recognition", (Tr. W. Turski and J.D. Cowan), Chapter 3, Thompson Book Company, Inc., 1967 (Library of Congress number 66-29876).

- Meisel, William S., "Computer-Oriented Approaches to Pattern Recognition", Chapters 1, 2, 4; Mathematics in Science and Engineering Volume 83, Academic Press, 1972 (Library of Congress number 72-77852.

Theoretically, a two-class classification problem requires only one feature for its solution. That is, if it is really a two-class problem and if the right feature can be found. The bus versus non-bus decision is apparently a two-class problem and there are four secondary features so, ideally, there should be no great difficulty in finding an appropriate decision rule. However, it is worthwhile to look closely at the structure of the data before attempting to formulate a decision rule. One way of doing this is to plot the average signals for each test vehicle shown in Figure 17 and see how the data are clustered. Since each of the average signals is described by four coordinates $F_1$, $F_2$, $G_1$, $G_2$, we must use an artifice to plot them on graph paper.

Consider the overall average of all vehicles as the reference point in the four-dimensional space of secondary features. This point, plus the two points corresponding to any two of the average signals (say Flxible bus No. 1 and the semi), define a two-dimensional plane. In this plane, it is an easy task to determine the distances from the overall average to each average signal and the plane angle between these same lines. For each pair of average signals, we can thus determine their subtended angle as seen at the centroid. If we list all such angles (36 of them for 9 points), we can then find, for each point, the point that has the smallest angular separation. Figure 18 is a polar plot of the distances from the centroid and the minimal angles starting with Flxible bus No. 1 (B1). The smallest angle is with bus No. 2 (B2). The smallest angle for B2 (excluding the angle with B1) is with the moving van (MV). From there, the smallest angle is with the GMC bus (B3) and if we then return to B1 the closure error is small. If B2, B3 and MV are excluded, the cycle starting at B1 proceeds to semi (S), step van (SV), Volkswagen (VW),

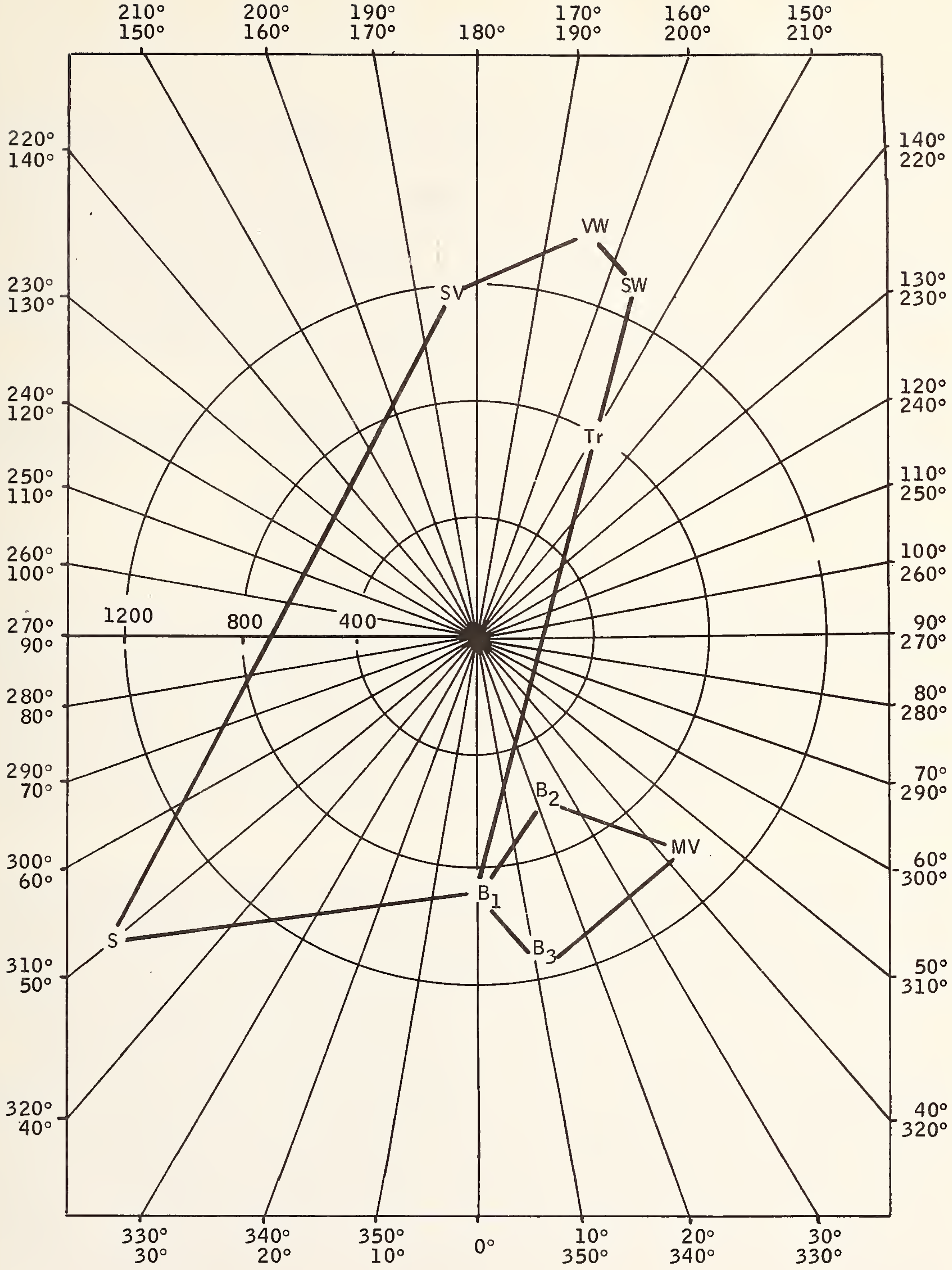


Figure 18. Polar Plot of Subclass Means

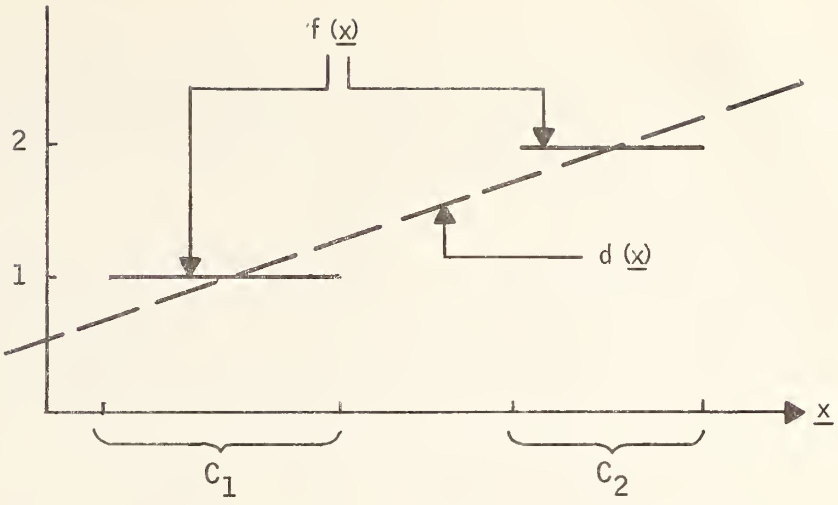
station wagon (SW), truck (Tr) and back to B1. This cycle has a very small closure error (i.e., the sum of the pairwise angles is very near 360 deg) indicating that these six points all lie near a common two-dimensional plane in the four-dimensional space. The cycle B1, B2, MV, B3 angles add to 0 deg and do not encircle the origin. This indicates only that these points are close together and it says nothing more about how they are placed. Figure 18 can thus be considered to be a reasonable representation of the relative locations of the nine average signals. From it we might conclude that, with the possible exception of the moving van, it should be easy to distinguish between the buses and other vehicles. On this basis, it was decided to proceed with the task of finding a decision rule.

The linear discriminant method provides the simplest means for describing a two-class decision rule. Let $\underline{x}$ denote the feature vector $(x_1, x_2, x_3, x_4)$. A linear discriminant is simply a linear function

$$d(\underline{x}) = \underline{a} \cdot \underline{x} + b = \sum_{j=1}^{4} a_j x_j + b.$$

A two-class decision rule can be stated: "Assign $\underline{x}$ to class 1 if $d(\underline{x}) \leq \theta$, otherwise assign $\underline{x}$ to class 2". Figure 19 illustrates one method of determining a useful linear discriminant function from experimental data. The points belonging to each class are specified and the least-squares fit determines the discriminant function. For our problem, the two classes could be any pair such as "GMC bus" and "station wagon" or "bus" and "non-bus". The discriminant function is determined from the experimental data for the chosen pair of classes. Its value can then be calculated for all of the experimental data and its ability to discriminate among the various data types can be determined.

Since the method used to determine the bus classifier is a non-standard technique, it may be of value to discuss both the method and its motivation in more depth. First, the motivation: The basic problem is a two-class problem (i.e., bus versus non-bus) but there are seven different kinds of vehicles in the data set used for the design (bus, semi-, moving van, etc.). Thus, in the worst possible case, the classifier would have to assign a given vehicle to one of seven possible classes on the basis of the signal measurements. Then a simple logical test would decide between tbe bus class and the six non-bus classes. However, Figure 18 indicates that the step van, Volkswagen, station wagon and dray truck are similar appearing in terms of the secondary feature values. These four kinds of vehicles can therefore be treated as a "cluster". Similarly, the three buses ($B_1$, $B_2$, $B_3$) form a cluster. We then have two clusters and the two remaining vehicle types (semi and moving van). The worst possible case is thus only a four-class problem in a four-dimensional space. Figure 18 shows one more piece of information that simplifies the problem even more -- the data are all located near a two-dimensional plane that lies in the four-dimensional space. Thus, ideally, there should



Let $\underline{x}$ denote feature values.

Define the characteristic function $f(\underline{x})$ as

$$f(\underline{x}) = 1 \text{ for } \underline{x} \text{ in class } C_1$$
$$= 2 \text{ for } \underline{x} \text{ in class } C_2$$

The discriminant function is that linear function $d(\underline{x}) = \underline{a}\underline{x} + b$ that minimizes the mean square error:

$$e(a, b) = \overline{[f(\underline{x}) - d(\underline{x})]^2}^{\,\underline{x} \in C_1, C_2}$$

Figure 19. Class Linear Discriminant Function

be two vectors that describe how the data are arranged (two vectors define a two-dimensional plane). These two vectors, of course, lie in the four-dimensional space and four coordinates (the secondary features) are needed to describe them. But once these vectors are found, the distribution of data can be described by two values, namely the distances along the two vectors. The problem is thus transformed from four-space to two-space and, as a result, becomes more tractable.

Second, the method: The first step in solving the problem is to find two vectors in four-space that describe the data. To do this, it is best to treat each vehicle as a separate class since this will preserve the most information about how the data are located in four-space. Ten vehicles were used in the data collection (three buses, seven others) and these were assigned numbers 1 through 10 to identify them. (The assignments are shown in Figure 20). These class-identifying numbers can be treated as values of some non-linear function in four-space that describes the data distribution and it is possible that portions of this function can be approximated linearly!

The solution method used was to find linear functions that approximate the significant parts of this function. Once the linear approximation is found, the gradient of the linear function describes the variation between the classes. (Figure 19 shows how this appears for two classes in one-space.)

To illustrate this method, Figure 20 shows the results of defining an identifying function that has the value = 1 for all busses and the value =2 for all non-busses. The best least-squares linear approximation to this function is given by $K_1$ as shown. When the value of $K_1$ is calculated for the secondary feature values obtained from busses, the values are found to lie in the range $0.67 \leq K_1 \leq 1.47$ and for the non-busses, the values were $1.47 \leq K_1 \leq 2.61$. (The ideal would be $K_1 = 1$ for the busses and $K_1 = 2$ for the non-busses.)

For Figure 21, the discriminant function $K_2$ was determined so as to have the value 1 for the bus samples and the value 4 for the semi. $K_1$ did a fair job of separating the busses from the top group in Figure 18 and $K_2$ was tried to get separation in the horizontal direction on Figure 18. The result is shown in Figure 21. $K_2$ does an excellent job in separating the busses from the semi, the step van and the moving van, but it fails for all the rest. Thus we have $K_1$ that can decide between bus and everything except the moving van and $K_2$ that is good on the moving van. Figure 22 indicates how these results can be combined to produce a decision rule that is valid for all cases. In Figure 22, the range of values for both $K_1$ and $K_2$ is plotted for each test vehicle. It is seen that the rectangular region $K_1 \leq 1.57$ and $K_2 \leq 2.89$ includes only bus samples and the non-bus samples are outside of this region. This provides the decision rule for distinguishing between busses and non-busses if the vehicle is a moving one.

A similar procedure was used to determine a decision rule for vehicles that stopped over the loop. In this case, only the normalized magnitudes $G_1$, $G_2$ were used. Again, two discriminant functions $H_1$ and $H_2$ were determined from tne experimental data. In this case, all of the data, from both the stopped and the moving vehicles, was used to determine the discriminant coefficients. This is valid because as far as the magnitudes are concerned, there is no difference between moving and stopped vehicles. The results of this work are shown in Figure 23. The moving van again is the problem. In this case, the moving van samples formed two clusters that had the bus classes between them. The decision rule was therefore formulated using two thresholds on each axis:

Bus if $-0.14 \leq H_1 \leq 2.4$ and $3.465 \leq H_2 \leq 5.465$.

Non-bus otherwise.

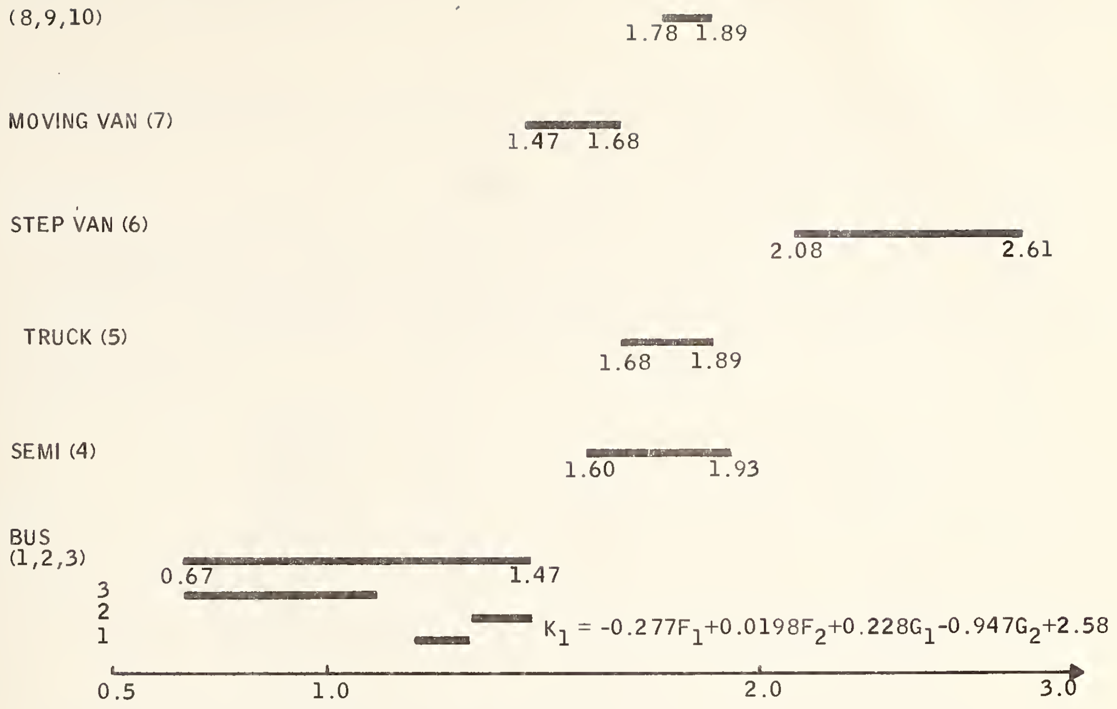


Figure 20. Discriminant Values for $C_1$ = Bus; $C_2$ Non-Bus

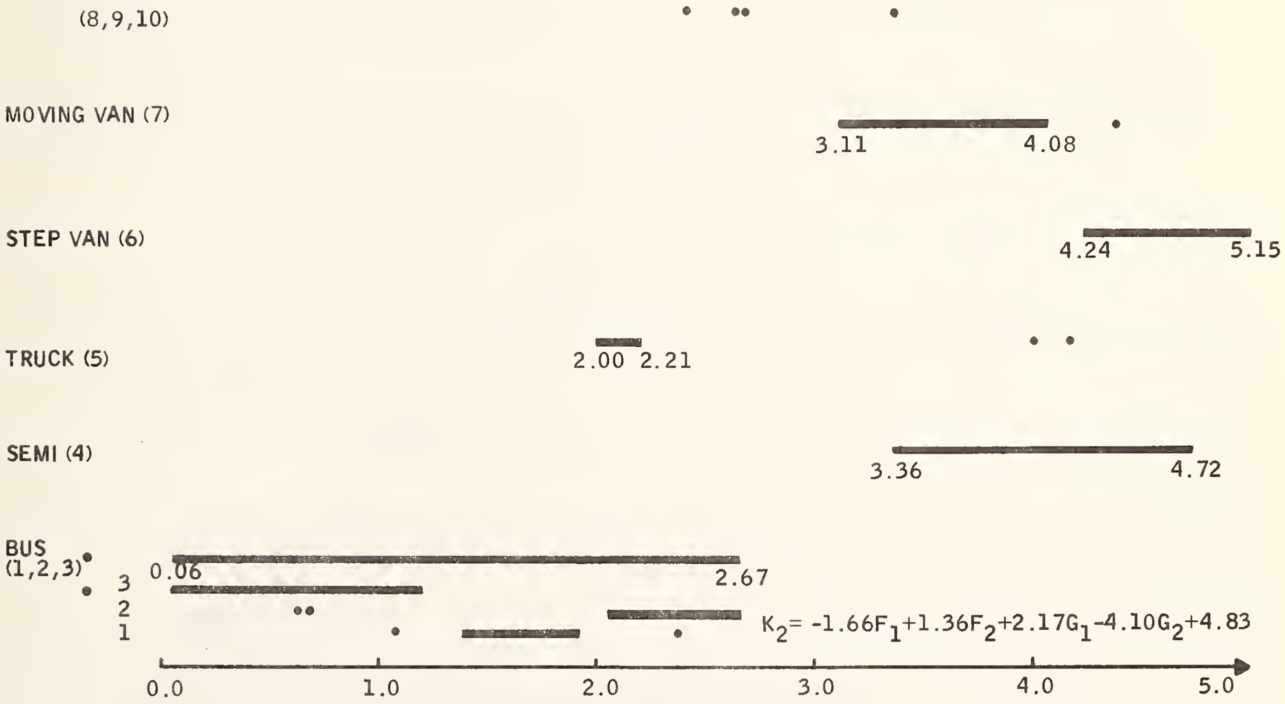


Figure 21. Discriminant Values for $C_1$ = Bus; $C_4$ = Semi

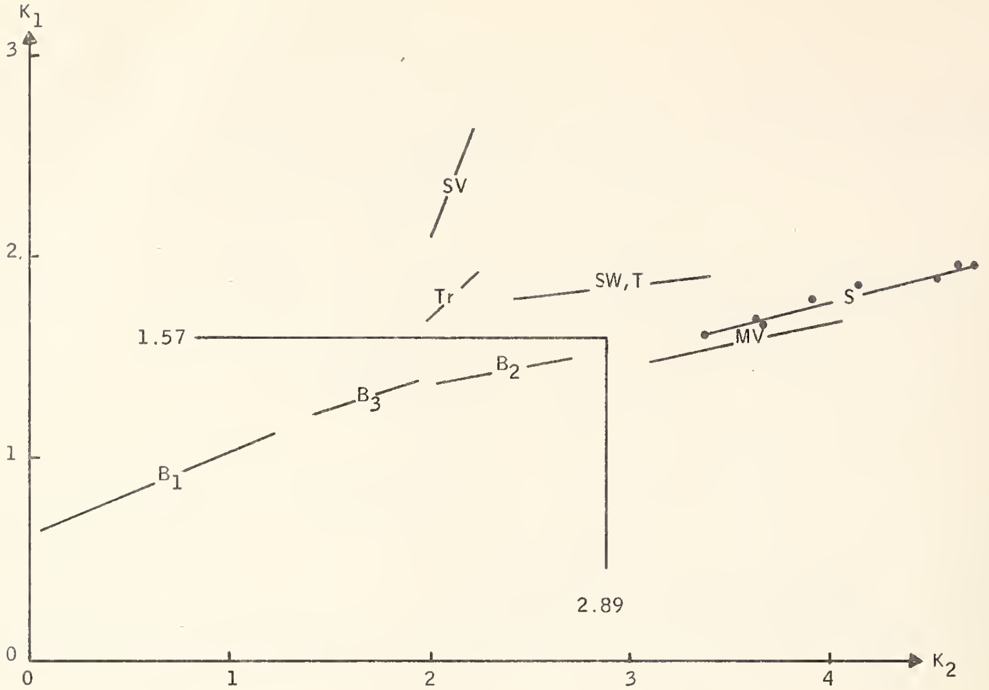


Figure 22. Joint Plot of Moving Vehicle Discriminant Values

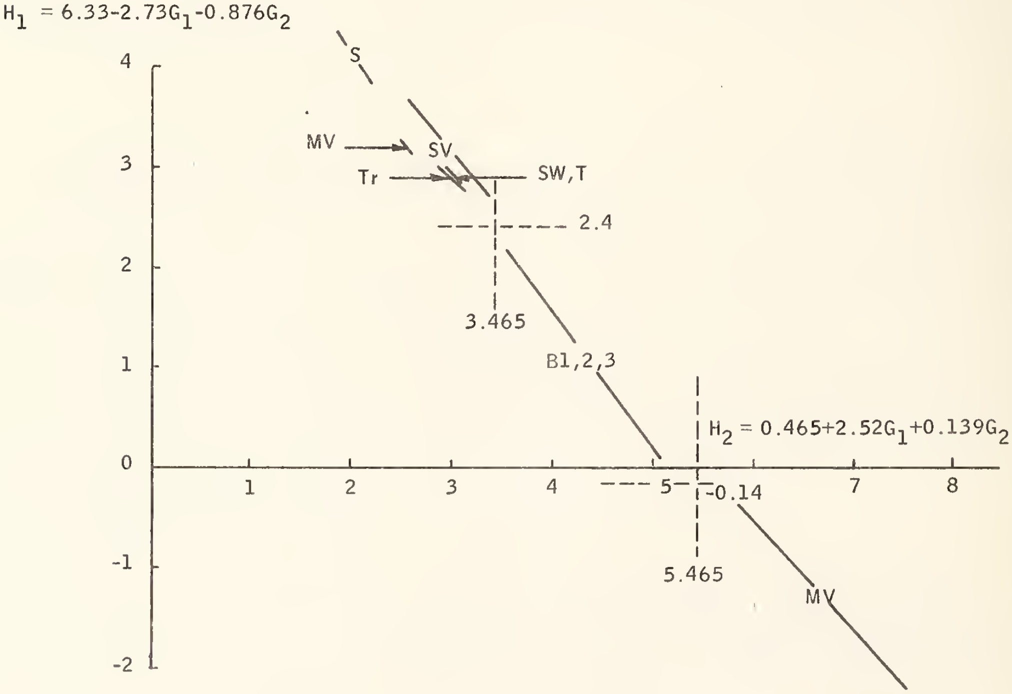


Figure 23. Joint Plot of Stopped Vehicle Discriminant Values

The decision rules for stopped and for moving vehicles could be determined independent of the method used to decide whether a vehicle was a stopped vehicle or a moving vehicle. However, the decision has to be based upon the primary or secondary features if it is to be included in the bus classifier algorithm.

If a vehicle stops while crossing the loop, then one or more of the primary time features $T_o$, $T_1$, $T_S$, $T_L$ will become large (see Figure 16). The normalized time features $E_1 = \frac{T_o}{T_o + T_L}$ and $E_2 = \frac{T_1 + T_S}{T_o + T_L}$ will behave in distinctive ways that depend on which of the times $T_i$ is large. Thus, if $T_L$ becomes very large, then both $E_1$ and $E_2$ will become small. The other possibilities are:

| Large Time Value | $E_1$ | $E_2$ |
|---|---|---|
| $T_o$ | Large | Small |
| $T_1$ | --- | Large |
| $T_S$ | --- | Large |
| $T_L$ | Small | Small |

Thus, we should expect moving vehicles to have small values of $E_2$ and intermediate values for $E_1$.

Study th the distribution of $E_1$ and $E_2$ for the experimental data led to defining small $E_2$ as $E_2 < 2.4$ and intermediate $E_1$ as $0.43 \le E_1 < 0.70$.

To determine a decision rule based on the experimental data, define two logical variables $L_1$, $L_2$ where

$L_1$ is true when $0.43 \le E_1 < 0.70$

$L_2$ is true when $2.5 \le E_2$.

The values (true or false) of $L_1$ and $L_2$ determined from the test data are:

| | | | | |
|---|---|---|---|---|
| $L_1$ = | T | T | F | F |
| $L_2$ = | T | F | F | T |
| Number of Moving Vehicles | 7 | 79 | 3 | 0 |
| Number of Stopped Vehicles | 16 | 2 | 8 | 3 |

Thus, it is clear that $L_1$ true and $L_2$ false is the condition to use for deciding that a vehicle was moving.

Figure 24 summarizes the various equations involved in the complete decision algorithm. The numerical values given were determined from data recorded at the Minneapolis test pad and included a variety of gain settings, both reinforced and nonreinforced surfaces. However, only two brands of buses were tested, Flxible and GMC. The complete bus classifier, shown in Figure 14 and using the equations in Figure 24, was simulated on the PACER 200 hybrid computer. The FM tape recordings of loop detector output from test vehicle runs were played at real time into the simulation to test its operation. Table 1 is a summary of the performance. In brief, there were no errors when the classifier was tested against the data from which it was developed.

Subsequently, additional data runs were collected at the Fairbanks Research Station in Washington, D. C. and at the corner of 18th and Pennsylvania during rush hour. The controlled runs were 23 passes (19 moving, 4 stopping) by an AMC bus. This brand of bus had not been included in the training data and the classifier missed the four stopped cases and one of the moving cases. (At this time, the gross decision test, shown in Figure 14, was a two-sided test $N_1 \leq N \leq N_2$ on the number of peaks N. The nominal limits were $N_1 = 2$, $N_2 = 6$. It was found that the stopped vehicles were being missed because the signals contained large spikes that were being detected as peak values. The spikes were probably due to the brakes being pumped. The problem was solved by changing the gross decision test to a one-sided test $N_1 \leq N$. Since only the first two and the final peaks are used, the existence of other peaks between the second and first makes no difference.)

The street data contained 18 bus signals within ± 4 feet of centerline and 439 non-bus signals. (The non-bus signals included four tour busses and one charter bus.) When this data was processed by the hybrid simulation, three of the bus runs were missed (two of these were AMC busses) but there were no errors in the non-bus runs.

After analysis of the entire data set, the thresholds for the moving vehicle classifier were changed so that the equations (see Figure 24) read

$$K_1 \text{ true if } \ldots \geq -0.960$$

$$K_2 \text{ true if } \ldots \geq -1.444$$

With this change, all of the moving vehicle runs (Minneapolis and Washington) were classified correctly.

With these modifications, the classifier simulation classified all 644 recorded test signatures correctly with the exception of two questionable cases where the identity of the vehicle is not certain and one very old bus that is sometimes classified as a non-bus. Estimates of false alarm and error rates cannot be obtained without testing a much larger data base.

$L_1$, . . . , $H_2$ are the logical variables defined in the preceding discussions.

Stopped Vehicle Test

$L_1$ true when $0.43 \leq E_1 < 0.70$

$L_2$ true when $2.5 \leq E_2$

A non-stopped vehicle is indicated when $L_1$ is true and $L_2$ is false.

Moving Vehicle Classifier

$K_1$ true when

$$-0.277F_1 + 0.0198F_2 + 0.228G_1 - 0.947G_2 \geq -1.01$$

$K_2$ true when

$$-1.66F_1 + 1.36F_2 + 2.17G_1 - 4.10G_2 \geq -1.944$$

A "bus" is indicated when $K_1$ is false and $K_2$ is false; otherwise a "non-bus" is indicated.

Stopped Vehicle Classifier

$H_1$ is true when

$$3.93 \leq 2.73G_1 + 0.876G_2 \leq 6.47$$

$H_2$ is true when

$$3.00 \leq 2.52G_1 + 0.139G_2 \leq 5.00$$

A "bus" is indicated when $H_1$ is true and $H_2$ is true; otherwise, the vehicle is a "non-bus".

Figure 24. Bus Classifier Decision Algorithm

Table 1. Bus Detector Simulation Performance

| Vehicle | Number of Runs | Number Detected | Passed Gross Test | Number Classified | |
|---|---|---|---|---|---|
| | | | | Bus | Non-bus |
| Motorcycle | 9 | 0 | --- | --- | --- |
| Bicycle | 6 | 5 | 0 | --- | --- |
| Flxible Bus | | | | | |
| ≤ 4 ft | 6 | 6 | 6 | 6 | 0 |
| > 4 ft | 4 | 4 | 1 | 1 | 0 |
| Volkswagen | 10 | 10 | 0 | --- | --- |
| Torino | 8 | 8 | 1 | 0 | 1 |
| Station Wagon | 9 | 9 | 2 | 0 | 2 |
| Semi | 9 | 9 | 3 | 0 | 3 |
| Truck | 11 | 11 | 3 | 0 | 3 |
| GMC Bus | | | | | |
| ≤ 4 ft | 13 | 13 | 13 | 13 | 0 |
| > 4 ft | 4 | 4 | 0 | --- | --- |
| Moving Van | 17 | 17 | 6 | 0 | 6 |
| Step Van | 58 | 58 | 4 | 0 | 4 |
| Totals | | | | | |
| ≤ 4 ft | 19 | 19 | 19 | 19 | 0 |
| > 4 ft | 8 | 8 | 1 | 1 | 0 |
| Non-Bus | 137 | 138 | 19 | 0 | 19 |

## Areas of Potential Concern

Classifier Coefficients -- Geometrically speaking, the classifier coefficients describe the location and normal direction of planes that separate the bus feature values from the non-bus feature values. The feature values are calculated from measured peak times and amplitudes. These, in turn, depend upon the vehicle detection threshold, the peak tolerance value and the bias estimating filter. These were described in the preceding pages but there is a certain amount of arbitrariness in the whole procedure.

The threshold, tolerance, and bias estimate were defined and simulated on the PACER and experiments were performed (using recorded signals) to determine good values for these parameters. Optimization was not possible because the optimum is the set that gives the best classification. Instead, the classifier is determined from the somewhat arbitrary choice of good values. Next, the features were defined by postulating a number of ways of combining measurements to overcome the stopped vehicle, different speeds, displacement from center line, and other variables that were apparent. Four of the candidate features were chosen because they had good statistical properties. Again, the choice was not a strict optimal one. Finally, the classifier coefficients were determined and the unpredictable result was perfect separation. This result produces one more arbitrary factor, because the slope of the planes can be changed slightly or they can be shifted back and forth a small distance and the classification is still 100 percent. Ideally, we should now return to the starting point and optimize the detection threshold, the peak tolerance, and all the other arbitrarily chosen values so as to produce the best classifier. And as more data become available the loop should be recycled to further improve the results. In other words, the ideal design process does not end untilt he optimum is realized, which may be never.

Thus, we must conclude that some arbitrariness is unavoidable if we are to be practical. But we must also decide whether the current level of arbitrariness is acceptable. We can only answer on the basis of data, experience and intuition, and the answer at this time is "yes"! However, we may learn otherwise as our data and experience changes! Fortunately, the digital implementation allows such changes to be made relatively easily, so the decision to accept current level information is not as critical as for a hard-wired device.

Vehicle Detection Threshold -- The threshold was chosen before the classifier structure and the kinds of features were known. The primary criterion, at that time, was to detect all busses. A secondary criterion was to detect enough other vehicles so that a representative set of non-bus signatures could be obtained for the classifier training. Later on, the threshold became important in another way; it determines the first and last signal intervals, that in turn are used to normalize the time features to reduce their dependence on vehicle speed. After the classifier had been simulated on the PACER, the threshold was tested by varying it over the 50 percent to 200 percent range to see if it was a critical value. Over this range, it was not, so the value was accepted.

Months later, when analyzing the Washington Street data with a simulation that now included the quantized values of time, amplitude and bias correction, it was seen that signal gain, quantization, bias correction and adjacent lane vehicles can couple with the threshold to produce errors. The effect can be counteracted for the existing data recordings by changing the stopped vehicle test from $E_2 > 2.5$ to $E_2 > 2.3$. This corrects the one bus that experiences this unique set of conditions and does not change the classification of any other vehicle. (It should be noted that this one bus has been run ten times on the simulation using different bias correction time constants and it was classified as a bus six times. The fundamental random variable appears to be the time of day at which the data is sampled! )

The change in the $E_2$ test value is almost insignificant, but it shows a number of things. First, it is an example of the arbitrariness mentioned earlier. The data available did not require or allow a more accurate determination of this value. Second, the change in the $E_2$ test value does cause some moving busses to be processed by what has been called the stop bus classifier. The busses that are affected are ones that are accelerating or decelerating. The "stopped bus classifier" thus becomes the "nonuniform velocity bus classifier". In retrospect this is much more reasonable - more data, better definition. The velocity estimate is not changed in any way by this. Third, we have an effect caused by the interaction of a number of "real-world" factors that was not seen when the parameters were varied one at a time in the simulation.

Fourth, it points out a way in which the chosen threshold value is very sub-optimal. This is discussed in the following paragraph.

Figure 25 illustrates the leading edge of a vehicle signature. The values $a_1$, $a_2$, $a_3$ are amplitudes (phase values) above the zero or steady-state bias level and the $t_1$ are the corresponding times.

If $a_2$ were the chosen threshold value and the intervals between the $a_1$ values is the amplitude quantization, then relatively large time errors are caused by the minimal (±) quantication unit change in the bias estimate. Figure 26 shows how the time value is made insensitive to bias estimate errors simply by using a larger vehicle detection threshold that places the threshold in the linear, large slope part of the signal preceding the first peak.

Similar coupling of the start (and stop) time exists with center line displacement, lead length, asphalt versus reinforced concrete, depth of paving material, loop age and any other factor that affects the signal peak phase value. Using a larger vehicle detection threshold would thus reduce the statistical variance of the vehicle classes and increase the confidence in the classifier. Unfortunately, changing the threshold changes the time-normalizing factor in a non-linear

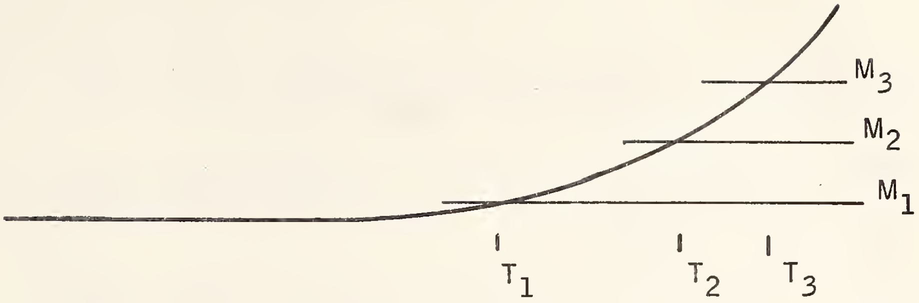


Figure 25. Leading Edge of a Vehicle Signature

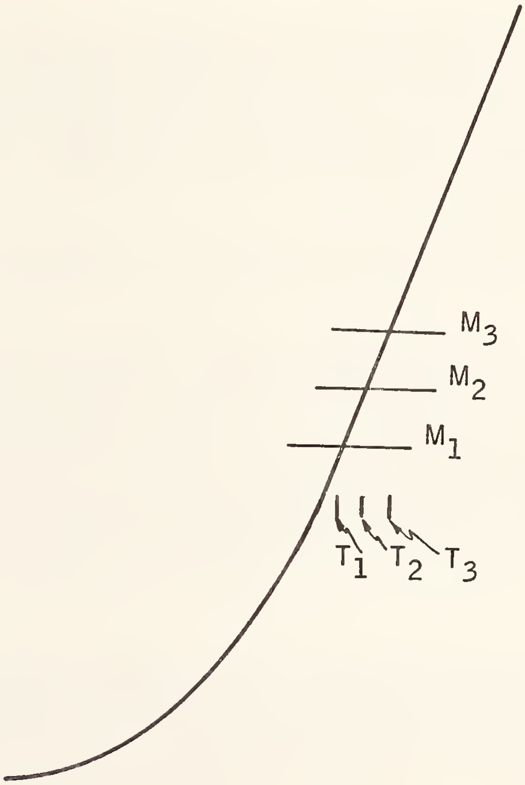


Figure 26. Magnitude versus Time

manner and this affects the decision boundary planes; i.e., a completely new set of classifier coefficients is needed, but the structure is the same. Since all of this analysis is the result of one exceptional signal, there is not enough evidence to justify iterating the design loop at this time. But, if the problem becomes a large one, the following steps can be taken:

1) Run the recorded tapes on the simulation using larger detection threshold values and doing quantized arithmetic. List bias and all features.

2) Perform graphical analysis of $T_i$, $A_i$, $E_i$, $F_i$, $G_i$, $H_i$, $K_i$ variation with the detection threshold.

3) Choose a value for the detection threshold that lies well within the linear part of the signal and which detects all buses.

4) If the graphical analysis in 2) shows a functional or highly correlated relationship between the threshold and the feature values, then use this to determine the new classifier coefficients. If the graphical analysis shows no simplifying relationship, then retrain the classifier using the new feature values.

5) Test the new classifier coefficients using recorded data tapes.

Bias Estimation -- When no vehicle is present the output of the loop phase detector is some nonzero constant value. This is the bias. The bias is expected to change slowly with time, temperature, etc., and it must be removed from the signal value before vehicle detection or feature extraction are done. The normalized amplitude features are insensitive to bias estimate errors, but the normalized time features are quite sensitive as discussed above. The sequel will summarize our experience with the bias and with ways of estimating it and then discuss some of the problems that may arise.

The expected total range of bias values is less than 8 deg (i.e., if the loop were initially calibrated to output 0 deg with no vehicle present, then -4 deg to +4 deg will contain all future bias values). The initial simulation estimated the bias with a digital recursive low-pass filter that operated only when no vehicle was present (i.e., signal minus bias less than detection threshold). Time constants in the range 1/10 sec - 10 sec were satisfactory. This algorithm involves a multiplication, which the preprocessor cannot perform. A second algorithm using shifts to replace the multiply was tested for the preprpcessor implementation. This did not work because the 12-bit preprocessor word was too small. We next tried a method that adds 1 to the bias estimate when the nonvehicle signal is greater than the estimate and subtracts 1 when the signal is less than the estimate. This is about as simple as a low-pass filter can be, and, when tested, it worked.

The disadvantage of this up/down counter method of filtering lies in the interrelation between time constant and quantization. The time constant is set by the time interval between successive bias tests (the bias estimate is compared with the sampled signal value only once every N samples). A count of 1 up or down corresponds to the signal quantization value (~ 1/4 deg at present). Thus if the time interval is too small the filter may jump up and down rapidly, so the time interval is chosen large to make the output relatively smooth. But, since this filter does not smooth the values finer than the quantization level, an error in the bias estimate cannot be corrected for the full time interval nor is it compensated for by smoothing. This causes a problem when the bias correction time happens to occur when the signal is increasing due to an approaching vehicle but before the signal reaches the threshold. In this case the bias will be increased one quantization interval above its correct value and the time estimates will be in error, as discussed previously.

For a large bias correction time interval (2-4 sec), this situation is infrequent since the increasing part of the signal is less than 0.2 sec long. But the error can occur another way. A vehicle in the adjacent lane or a lane straddler can cause a small increase in signal level that looks like a rapid bias increase. If this happens before a vehicle comes by in the main lane then the bias estimate is almost certain to be too large. Something of this happened in about 25 percent of the Washington street data. Using a larger vehicle detection threshold, as discussed in 2), would cure this problem by making the time features insensitive to bias errors. An alternate but less desirable way would be to introduce bias smoothing.

By using scaled values (stored value = $2^8$ x bias estimate) the smallest bias estimate increment can be made a fraction of the quantization interval. Thus an error would have to persist for a number of correction intervals before its effect would appear in the lowest significant position. To implement this in the preprpcessor without using the shift instructions:

Scaled bias estimate $\longrightarrow A_o$

add bias increment

$A_o \longrightarrow$ memory = new scaled bias estimate

$A_o$ bits 8 - 11 $\longrightarrow A_1$ = bias, unscaled

This would handle a 0 deg - 4 deg range of bias value, and could be expanded to 0 deg - 8 deg by using arithmetic overflow detection.

For the sake of simplicity of implementation and because the simulation was relatively insensitive to the following method of bias estimation, a simple scheme of bias correction was employed. At present, the preprocessor adds or subtracts 1 from the bias estimate if the estimate is less than or greater than the nonvehicle signature value. The estimate is not change if it is equal to the nonvehicle signature value. This update process occurs once every 1.5 seconds.

## FM Loop Detector Analysis

Forty-five non-bus and 10 bus runs were made and recorded on analog tape. The analog tape was thenprocessed by the bus detector classifier hybrid simulation developed previously for the loop phase detector. With no changes in any of the simulation's parameters, the classification results were:

| | |
|---|---|
| Non-bus | No errors |
| Bus | 90 percent correct classification |

Because of the small size of this data set and because it contains no vehicles that tend to look like buses, this result must be treated as only a rough indication of how the classifier can perform when using the FM loop detector.

The FM loop detector is an asynchronous device. Successive output values occur at time intervals that depend upon the instantaneous frequency of the loop oscillator. The D/A converter is a ladder network that responds like a summing amplifier with each bit of the FM output word being a separate input. The recorded analog signal is thus a step waveform with the steps occurring at unequal intervals. The hybrid simulation also operates asynchronously and independently of the step intervals on the input tape. The digital sample inputs to the simulation can thus occur at any time during the interval corresponding to one output value from the FM detector. It is also possible that the simulation's sample is taken at the time when the waveform is changing from one step value to the next. In this case, the sampled voltage will be distorted by the transient characteristics of the D/A converter, the tape recorder and playback amplifiers and the PACER input amplifiers which potentially can produce signal values that are not representative of values that would appear in an all-digital device. These possible differences between the simulation and the actual implementation of the FM loop detector give another reason for treating the preliminary results as only indications of true performance.

To obtain maximum information about the FM loop detector from the small data set, tests were performed to determine the sensitivity of the classification to changes in the simulation's parameters. The first parameter is the change in signal sampling time produced by processing the recording a number of times. Since the computer and the tape recorder are not synchronized, the time points at which the signal is sampled depends upon precise position of the analog tape at the time the first sample is made and this is a random variable. Hopefully, the classifier would not be affected by this random variable, since this is equivalent to the absolute time at which a vehicle arrives at the loop and it would be bad to have the classifier depend on the time of day. The feature values and classifications for the phase-loop detector are not affected by the starting time; however, the FM loop data when processed four times yielded zero errors once, one error twice, and two errors once. Even on the vehicles that were classified correctly, the feature values differed between trials; and of the 10 moving bus recordings, five of them were called "stopped bus" at least once during the four processing trials.

The other parameter that was varied was the "peak tolerance theshold." This value determines when a peak occurs. For the loop phase detector, this parameter was tested over a 50 percent to 200 percent of nominal range without affecting the classifier. For the FM loop detector, the tolerance must be considered to be an integer multiple of the data quantization, (q = signal change that produces a change of 1 bit in the detector output). The starting time tests were done with tolerance = q. Two additional tests were run with tolerance = 2q and 3q. For the 2q case, there were no errors and for the 3q case, there was one error. With a test size of 1, we can conclude that q and 2q yield the same result, but for 3q, the error was caused by failure to detect the bus at the gross test. One peak was too small to be seen at this tolerance level. Thus, we are confronted with the smallest significant signal change being only twice as large as the smallest observable change. This appears to be a critical point.

Until more experiments are done, we can only conjecture about the way to resolve these problems. It is possible that the starting time dependence is a result of the D/A transient response coupled with asynchronous sampling. If so, then the simulation must be changed to assure that the step values are sampled only at steady state, thus making the simulation agree with how the actual device would operate. Alternatively, it may be that the starting time dependence is related to the quantization/tolerance interaction. If further investigation confirms that the quantization and tolerance are critically related, then we will have to consider increasing the 1 MHz oscillator frequency (Figure 33) and/or increasing the number of bits in the output register. The latter would impact the preprocessor design by making multiple precision operations or larger word size necessary.

In conclusion, examination of one small data set indicates that the FM loop detector has the same classification potential as the loop phase detector, but it has unique potential difficulties that will require more data collection and analysis to resolve.

### Bus Priority Control Unit - Software

The software developed for the passive bus detector system is documented in Volume II of this report, which contains Appendices A, B, and C.

### FORTRAN IV Version of the Vehicle Classification Algorithm

This program duplicates the logic and arithmetic operations of the vehicle classification algorithm which operates in the microcomputer unit of the BPCU. This progra m is written in FORTRAN IV for operation on a general-purpose computer. Using this program, the vehicle classification can be analyzed offline with respect to the classification or rejection of particular vehicle signatures.

See Appendix A, Volume II, of this report.

### BPCU Program for Minneapolis Installation

This program is written for the field test of the Passive Bus Detector system in Minneapolis. The detector configuration consists of a pair of detectors (upstream pre-empt, downstream clearance) on opposing approaches along the main street. No detectors are placed on the cross street.

The pre-empt operation includes both green extension on the main street and green advance by contraction of the cross-street green.

The timing plan for the traffic signals consists of two sequences of intervals. Traffic flows both directions on both streets. Opposite directions are controlled identically. There is no special signalization for the buses or for turning movements.

See Appendix B, Volume II, of this report.

### BPCU Program for Washington Installation

This program is written for the field test of the Passive Bus Detector system in Washington, D.C. The detector configuration consists of one detector in a curbside, reserved bus lane on the main street. The cross street access from one side makes a T intersection.

The pre-empt operation involves the insertion of a sequence of intervals on the special bus signalization and stopping of the parallel, same way traffic on the main street. The bus is detected and classified downstream of the passenger loading zone and upstream of the stop bar. The exclusive green for the bus may be extended incrementally for each additional bus during the timing of that interval.

A bus classification during the cross street sequence causes the insertion of the bus sequence in the next cycle.

The timing plan for the traffic signals consists of three sequences in addition to the bus sequence which replaces intervals at the beginning of the main street sequence. Traffic flows both directions on the main street and the cross street. The southbound flow on main street has a left turn signalization for entrance to the cross street. The bus has special signalization for its exclusive green interval.

See Appendix C, Volume II, of this report.

## Bus Priority Control Unit - Hardware

### Introduction

The Bus Priority Control Unit (BPCU) consists of the bus detector and intersection control hardware exclusive of the inductive loop electronics. Figure 7, in Section II, is a functional block diagram of the BPCU that shows, with the exception of the power supplies, the major electronic subsystems. The operation of each subsystem is discussed in detail in the pages that follow. Figure 8, also in Section II, is the completed assembly.

### Loop Electronics

Having chosen the inductive-loop transducer for the bus detector, we had to settle on a loop-detector design. The hardware had to yield the same vehicle signatures as were gathered using instrumentation. In order to understand the loop-detector electronics a little better, we will discuss briefly the theory of operation of the inductive loop.

Figure 27 is a lumped-constant schematic diagram of an ILD transducer section. The capacitance, C, consists of the loop lead-in capacitance plus an additional discrete capacitance in the detector electronics which is required to make this circuit a parallel-tuned circuit. The inductance, L, is made up of the lead-in inductance plus the loop in the roadway. The circuit resistance, R, consists of the DC resistance, plus skin-effect resistance, of the lead-in and the loop. Although these parameters are made up of fixed and distributed impedances, a lumped-constant model such as Figure 27 is a reasonable first-order representation of the real world.

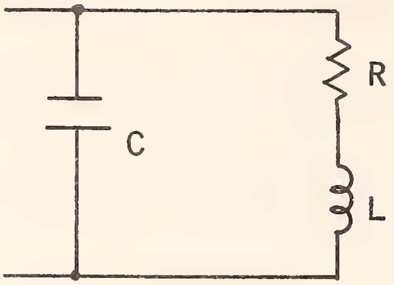


Figure 27. Inductive-Loop Transducer Circuit

The principle of operation is that when a vehicle passes over the loop, its inductance changes. For a three-turn, 6-ft x 6-ft loop of 72 μH, this change is typically 0.23 percent for a motorcycle to 8 percent for an automobile.[1] The tuned circuit is thus detuned from its resonant condition. The task of the detector electronics is to detect this detuning.

Detuning of a tank circuit can be detected in three ways: (1) the voltage change across the tank can be detected; (2) the phase shift between the driving source and the tank oscillation can be detected; and (3) the new resonant frequency of the tank with the vehicle present can be determined. Of these, only methods 2 and 3 will be considered for further discussion. Most commercial loop detectors today use method 2. Method 1 will not be considered because the total magnitude change is very small when compared with the sensitivity obtainable via phase detection. Field tests at Honeywell have shown that the amplitude change must be amplified by 500 or more to achieve the same signal level as the phase-shift signal which is amplified by 7. The reason for this is that loop Q's are typically low (5 to 16). This indicates that higher S/N ratios can be achieved by methods other than amplitude detect:on.

The equations for determining resonance of the tuned circuit of Figure 27 are:

$$\omega_o = \sqrt{\left(\frac{1}{LC}\right) - \left(\frac{R}{L}\right)^2} \qquad (2)$$

and

$$\theta = \tan^{-1}\left(\frac{\omega C\,(R^2 + \omega^2 L^2) - \omega L}{R}\right) \qquad (3)$$

[1] Traffic Data Systems AN-302

These equations can be derived from the Y parameters of the circuit. If we develop these one step further by letting $Q = \omega_0 L/R$, we get

$$\omega_0 = \sqrt{\frac{Q^2}{CL\,(1 + Q^2)}} \tag{4}$$

and

$$\theta = \tan^{-1}\left\{Q\left[\omega^2\, CL\left(\frac{1}{Q^2} + 1\right) - 1\right]\right\} \tag{5}$$

Now we can use these equations to predict the phase and frequency response of a tank circuit which is detuned by vehicle presence.

If Q isheld constant while the loop inductance (excluding the lead-in) is changed, we get Figures 28 and 29. The parameter, D, is the lead-in length. Both Q and D are measured values from field tests at Honeywell. Table 2 lists the loop and lead-in parameters used to generate Figures 28 and 29.

Table 2. Loop and Lead-in Parameters

| Loop | | Lead-in | |
|---|---|---|---|
| Number of turns | 3 | Inductance | 0.22 µH/ft |
| Inductance | 72 µH | Capacitance | 24 pF/ft |

From Figures 28 and 29 the electronics designer can expect to see the range of values within which his circuit must operate.

Some concern existed during the program as to which parameter in the inductive-loop transducer should be operated upon by the detector electronics for best long-term performance, especially when one considered the effects of Q. A well-known fact is that as soon as a loop and its lead-in is installed, the Q begins to deteriorate due to insulation breakdown on the wire itself and/or deterioration of electrical connections in the loop circuit. Most loop detectors today require a loop Q of 5 or greater for good operation. The concern, then, was which parameter, frequency shift or phase shift, more tolerant of long-term changes in loop Q.

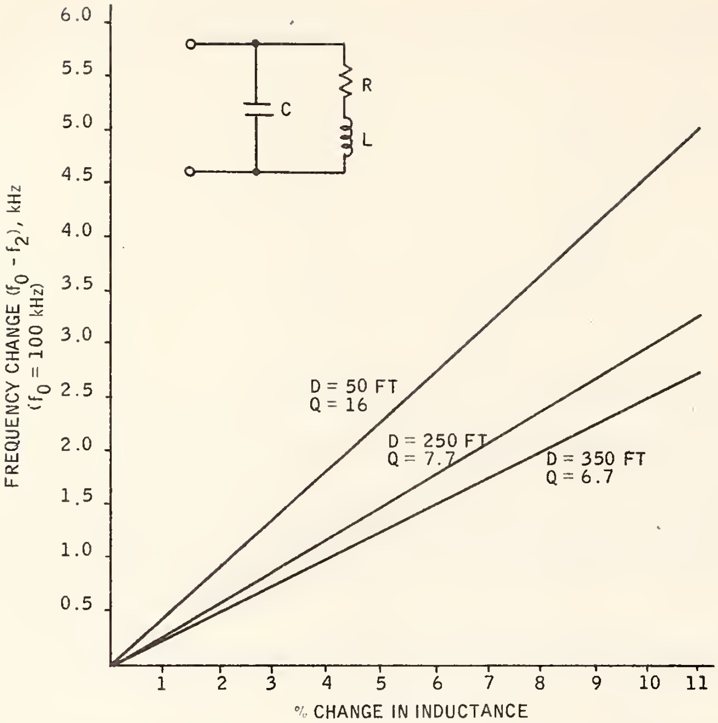


Figure 28. Transfer Characteristic, $F_o$ vs L

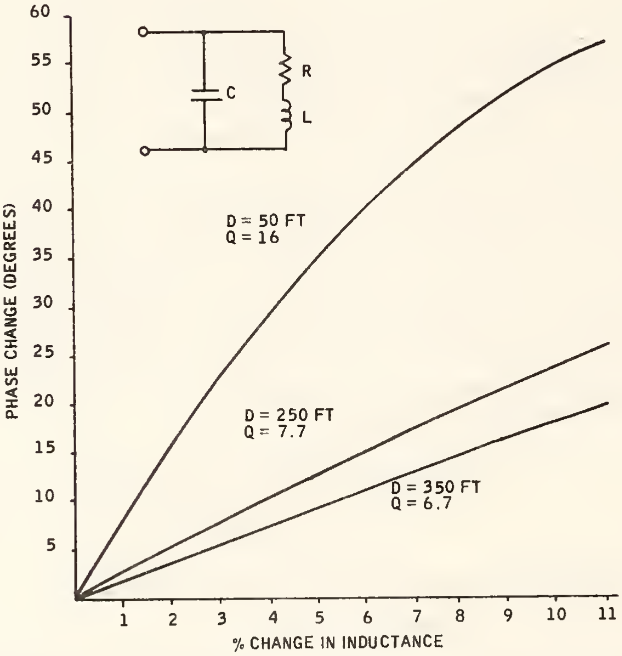


Figure 29. Transfer Characteristic, ϕ vs L

Equations (6) and (7) below, derived from Equations (2) and (3), were used to calculate the set of curves shown in Figures 30 and 31. These curves show the effect of Q upon phase and frequency shift for various given lead-in lengths, allowing Q to change with vehicle passage. These curves are based upon a percentage change in total inductance due to a vehicle. A scale-factor curve, Figure 32, is needed to convert percent total-inductance change to a percent loop-inductance change. One can readily see from Figures 30 and 31 that for Q > 5, a detector based upon measuring frequency shift (FM) has more constant output signal for a given percent inductance change as Q is changed. That is,

$$\omega_2^2 = \frac{\omega_o^2}{1+\alpha}\left[1 + \frac{\alpha}{1+\alpha}\left(\frac{1}{Q_o^2}\right)\right] \tag{6}$$

and

$$\tan\phi = -\alpha \frac{\left[1 - \alpha - \frac{1}{Q_o^2}\right]}{1 + 1/Q_o^2} Q_o \tag{7}$$

where $\alpha$ = incremental change in total inductance due to vehicle presence.

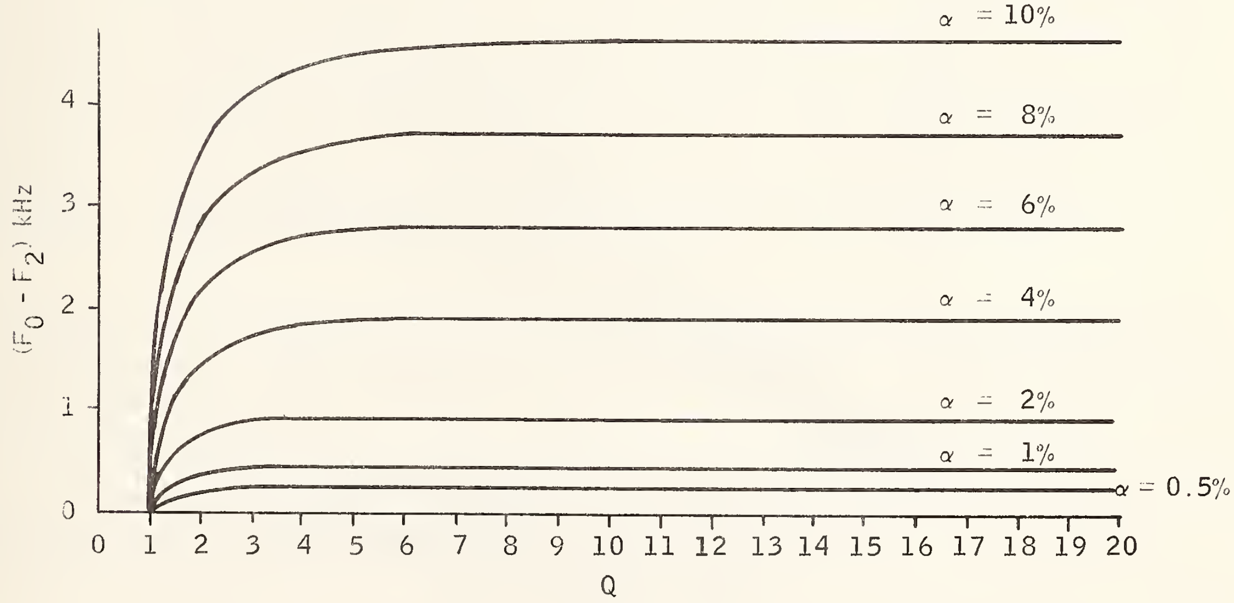


Figure 30. Frequency Change versus Q for $\alpha$ = Change in Total Inductance

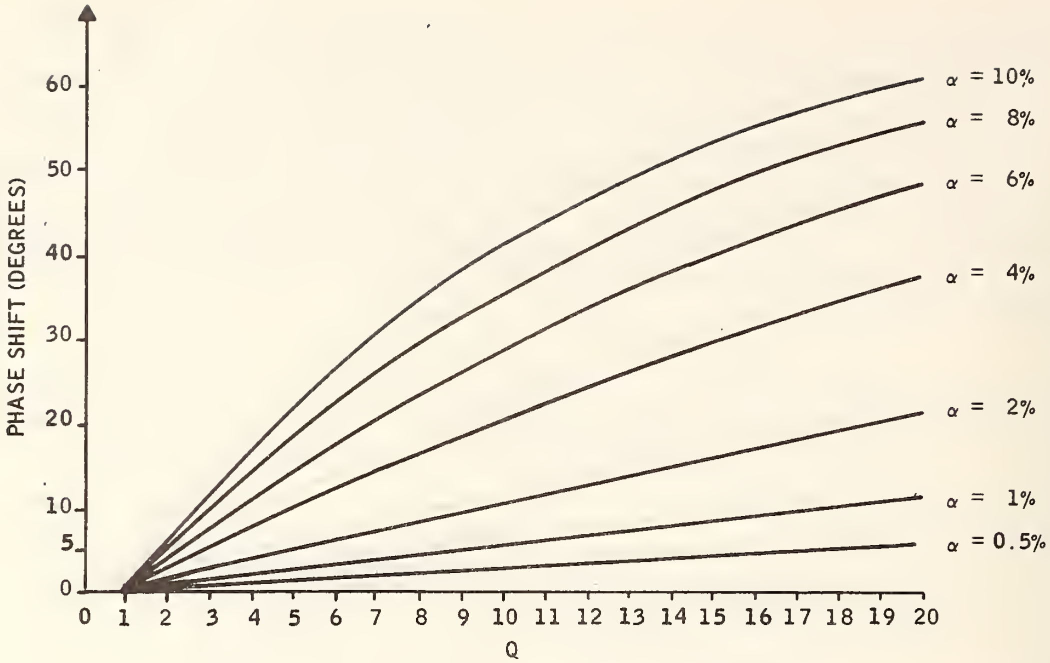


Figure 31. Phase Shift vs Q for α = Change in Total Inductance

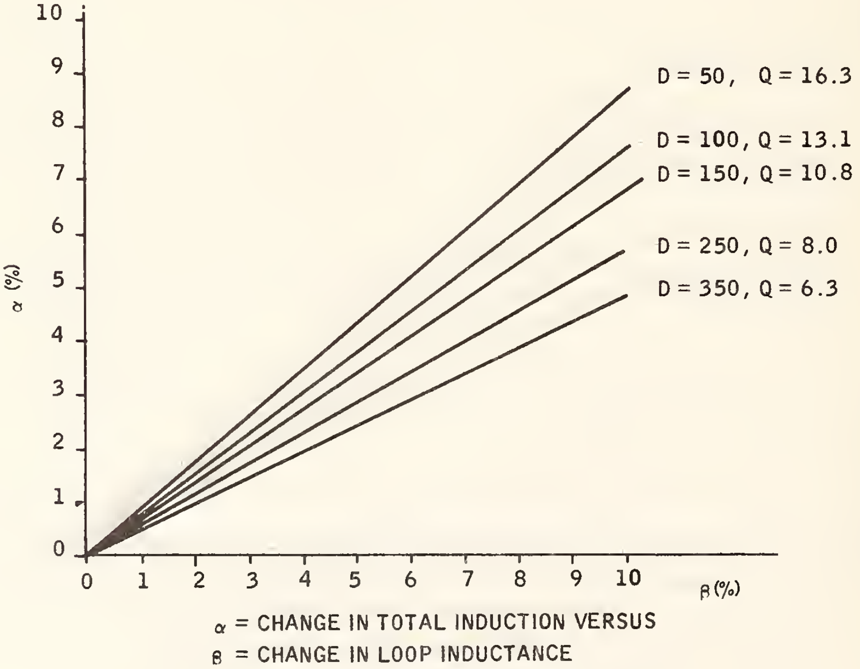


Figure 32. Scale Factor Curves

Although the foregoing analysis showed an FM loop detector to be superior to a phase-mode detector, with considering changes in Q, the loop detector design proceeded with the phase mode. The overriding reason for this was that the signature acquisition effort was nearly completed and not enough signature data existed from the FM mode detector. The bus detector classifier design, based on the phase mode, was also close to completion at this time. Also, insufficient project funds existed to evaluate and develop a new bus detector based on the FM mode. The decision was made, therefore, that the bus detector preprocessor should be designed to accept the FM-mode loop electronics, with little redesign, at a future date. Nonetheless, some limited amount of field data was taken on the FM mode.

The signature data base for the bus detector program (see Volume III) includes those taken with FM electronics which were designed especially for signature acquisition (see Figure 33). Runs 395 through 449 contain signatures of a Volkswagen, Step Van, Station Wagon, and a Transit Bus. The data log contains the detailed test conditions.

When runs 395-449 are compared with earlier runs under the same test conditions in which the phase response was recorded, one finds a striking similarity between the signatures. The same signature features are present in both cases. A discussion of how these data were analyzed is included in preceding sections of this report.

Figure 33 is a schematic of the FM-loop electronics. Q1 is the oscillator which drives and resonates with the loop. Its frequency is dependent upon the loop and lead-in impedance, and is designed to operate near 100 kHz. Amplifier A1 converts the sine-wave signal on the loop to a square wave, for digital processing. U8 is a period counter which counts 125 periods of oscillation of Q1. U1 and U2 are binary counters which function as a 12-bit counter, driven by the 1-MHz crystal oscillator. At the end of 125 Q1 periods, the counter (U1, U2) value is loaded into a 12-bit register (U3, U4, U5). Flip-flop U9 and associated gates control the above process. As the loop inductance changes, therefore, with the vehicle presence, the register value (U3, U4, U5) changes proportionally.

Now, in order to easily record the 12-bit register values, i.e., signature, the digital values must be converted to an analog signal. U6 and U7 are logic level converters which change the CMOS levels to TTL. U10 is a D/A converter. A2 is a summary/buffer amplifier for the D/A converter. A3 is a variable-gain amplifier.

The schematic diagram of the phase-loop electronics designed for the bus detector is located in Appendix III of this volume. Digital ICs are used wherever possible to minimize production costs.

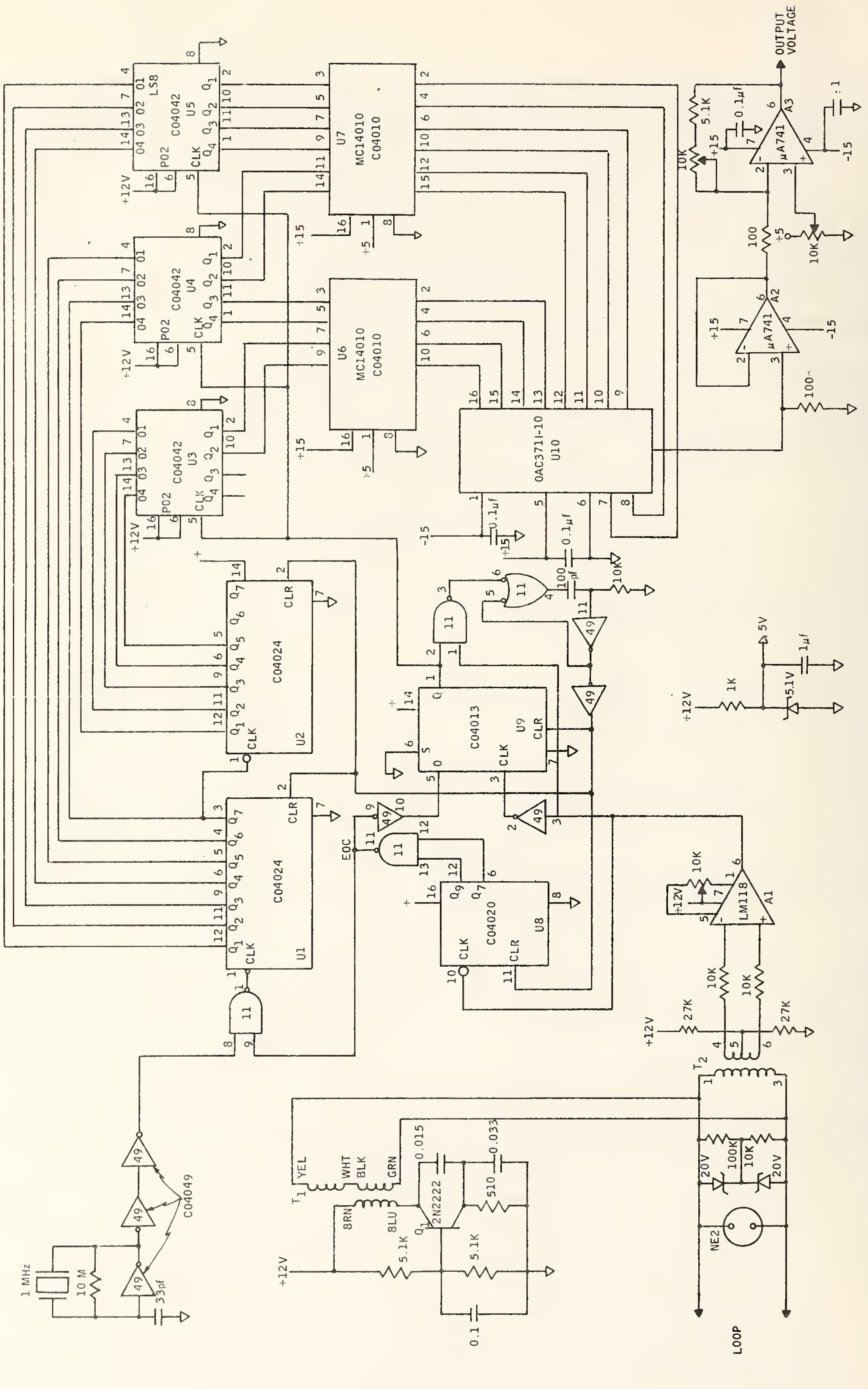


Figure 33. Digital Loop Driver and Detector

The floating loop is driven by a crystal oscillator-biphase generator-buffer circuit, U2. U2 is a CMOS, digital, hex-buffer IC. The vehicle signature is picked off the loop by transformer, T1, to get common-mode noise rejection. The loop oscillations are squared by amplifier A1 and sent to the phase detector, U1, where they are compared with the oscillator reference. U1 is a digital CMOS exclusive-OR IC and produces a pulse width which is proportional to the input phase difference. Amplifier A2 emplifies the phase difference produced by U1 and is integrated by the RC network. Switches S4, S5 and associated resisters provide variable-gain adjustment on A2. This is to compensate for lower signal levels in reinforced roadways and long lead-ins. U3 is another phase detector, used for tuning, which indicates which side of null the detector is tuned. Q1, Q2 and associated light-emitting diodes (LEDs) are used for tuning, as are switches S1-S3 and associated capacitors.

U4, U5, Q3 and associated LEDs are used by the preprocessor to sense the power-up, background initialization condition. When power is first applied, or switch S5 is pressed, flip-flop U3 is cleared. This is sensed by the preprocessor which performs a long-term background average and resets U5 when complete.

Power for the loop-detector circuits is supplied by two power supplies:

1. +5V - Acopian Model 5E50A (500 mA)
2. ±12V - Electrostatics, Inc. Model 10-1212 (350 mA)

The power supplies are arranged so that up to four loop detectors can be powered from one pair of supplies. Figure 34 is a photograph of the inductive-loop electronics.

Tuning procedures are contained in Appendix I of this volume.

## Preprocessor

The preprocessor (PP) is a data-reduction device whose purpose is to condense and operate upon real-time analog data from a minimum of 16 traffic detectors. In performing this function, the preprocessor acts as a front-end device for the bus classifier computer (microprocessor) (Figure 35).

The specific operations performed on each analog detector signal are: (1) determine the number of peak values in the vehicle signature, (2) measure the peak value(s) in the signature, (3) determine the time intervals between specific events in the signature, (4) transfer the above information to the MCU, and (5) perform long-term (background) signal averaging for use in threshold detection. In order to extract the above features in real-time from a large number of vehicle signatures, 16 minimum, the preprocessor was designed to be a microprogrammed, digital device. Performing feature

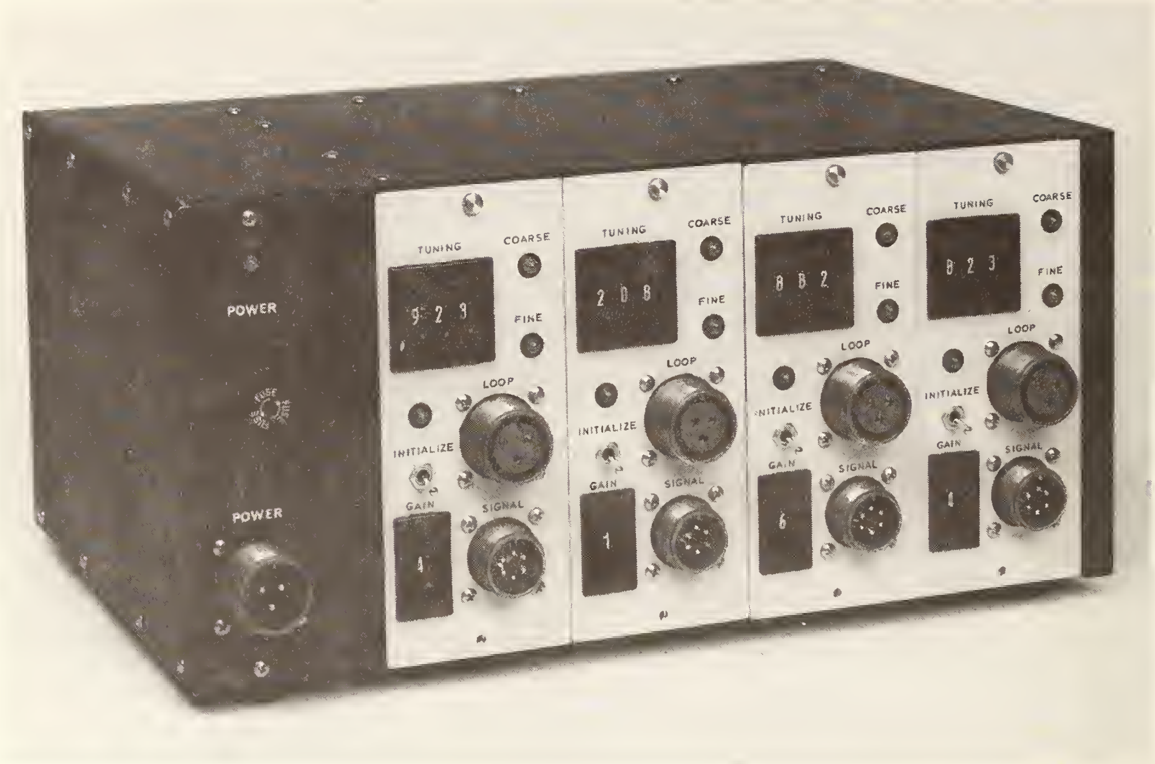


Figure 34. Inductive Loop Detector

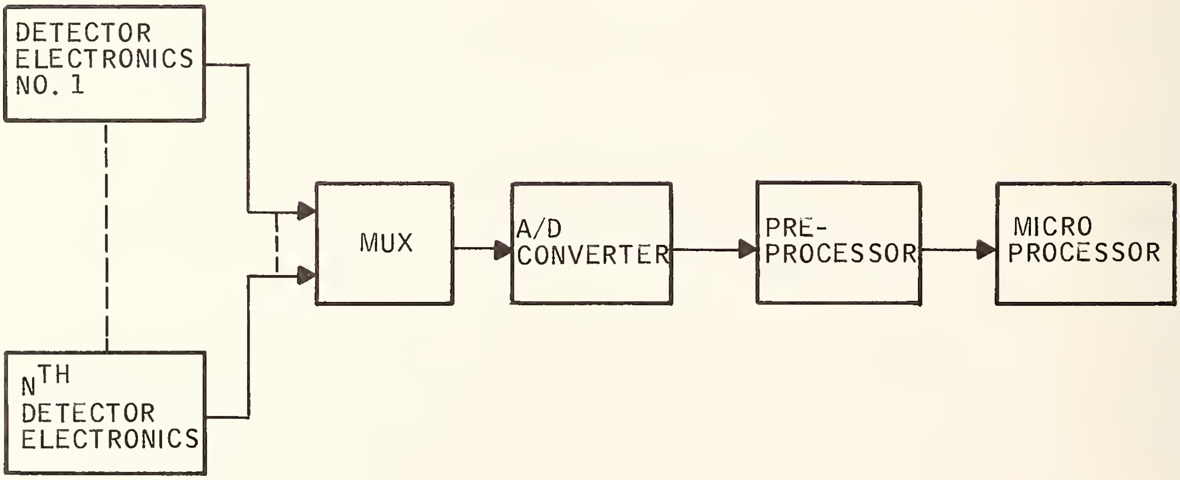


Figure 35. Bus Classifier Baseline Functional Block Diagram

extraction in a digital preprocessor, instead of analog, offers some very good advantages. For instance, a digital preprocessor can, up to a point, simultaneously process several vehicle signatures without adding hardware for each detector. Also, functional hardware parameters can be more closely defined and controlled in a digital processor. Thirdly, a digital preprocessor is less susceptible to temperature and component aging than an analog preprocessor. Finally, the performance of a digital preprocessor can be more consistent from unit to unit.

A side benefit of a digital processor is that it can perform a long-term (background) signal average for each detector without any hardware penalty. Performing the long-term signal average in software releases the detector electronics from having to track temperature variations of the transducer or track long-term component aging in the detector. This means that the detector electronics can be made much cheaper.

The analog multiplexer (MUX) and the A/D converter shown in Figure 35 are an integral part of the preprocessor and are located on the preprocessor circuit boards. Figure 36 is a flow diagram which shows the general process by which each and every vehicle signature is stripped of its significant features. In order to perform these functions, the preprocessor has the following general characteristics:

- Program memory: 256 words x 12 bits (PROM)
- Arithmetic logic unit: 12 bits wide
- Temporary memory storage: 256 words x 12 bits (RAM)
- Speed: 2-μsec machine cycle (instruction lookup plus execution)

Preprocessor Functional Description -- The basic building blocks of the preprocessor (Figure 37) are: (1) control memory, where the microprogram is stored; (2) $A_0$, $A_1$, B, and C registers (working registeres) where data are temporarily stored and manipulated; (3) arithmetic logic unit (ALU), where mathematical operations are performed; and (4) preprocessor memory, where individual detector data are stored. Of course, interspersed among these building blocks are other registers, gates, etc., which integrate the system as a whole.

An integral part of the preprocessor is the detector interface which multiplexes the signals from a maximum of 16 detectors. This interface will be described in detail in later sections.

General-Purpose Registers -- The preprocessor contains four general-purpose working registers: $A_0$, $A_1$, B, and C. Their functions are as follows:

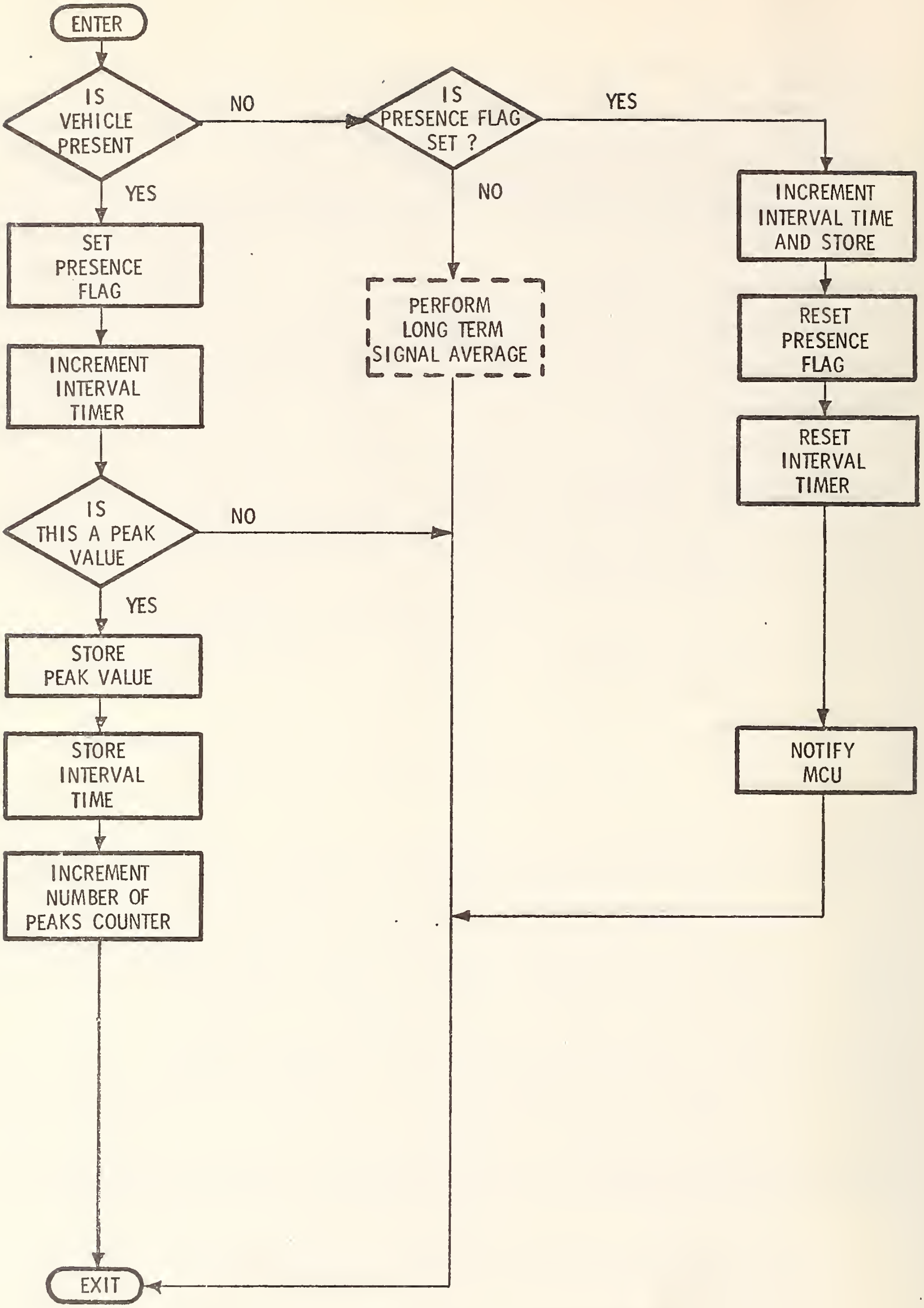


Figure 36. Preprocessor Flow Diagram

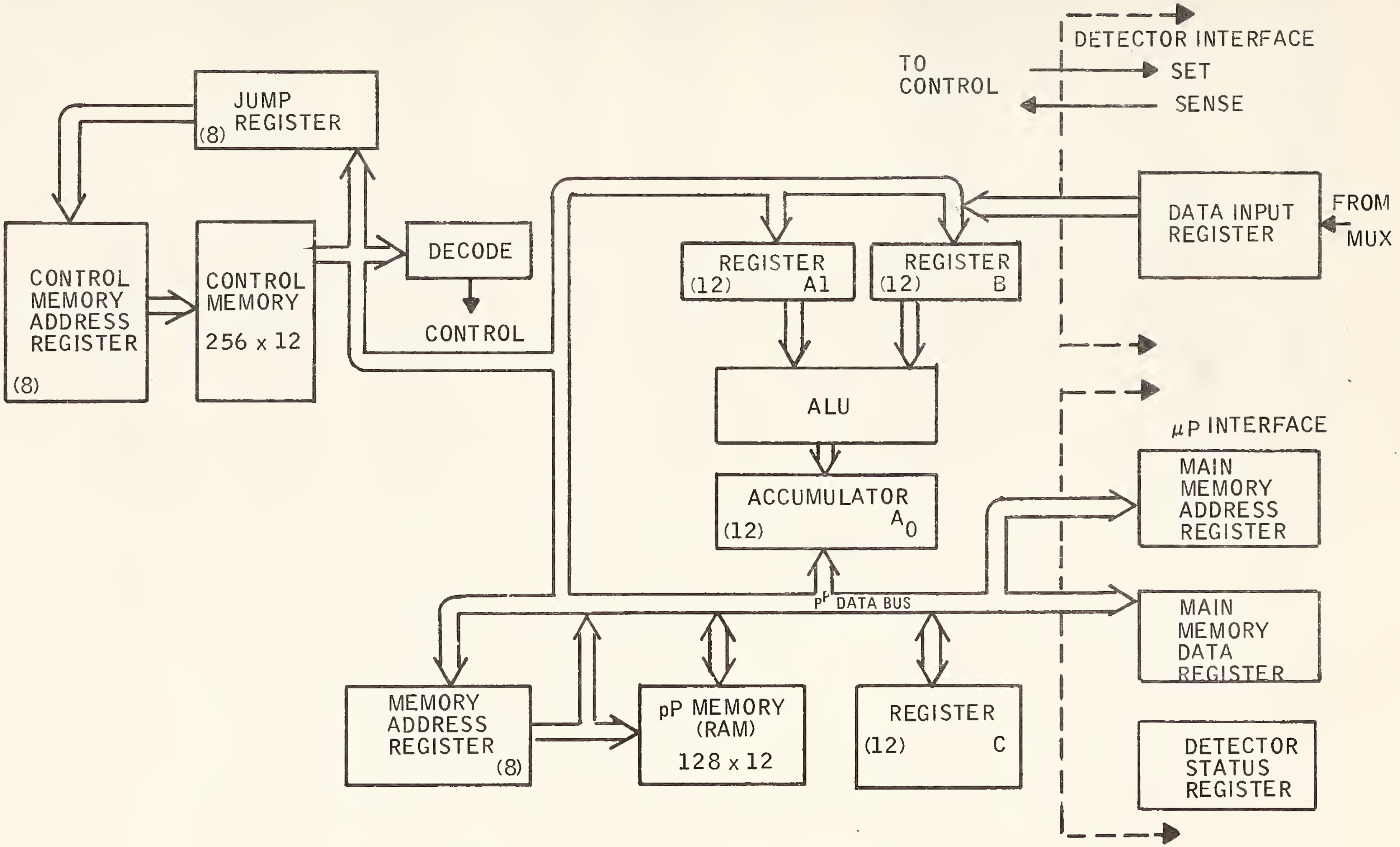


Figure 37. Preprocessor Architecture

- Register $A_0$ is designated the accumulator for arithmetic operations. It is 12 bits wide.

- Register $A_1$ is a 12-bit register which is intended for use in conjunction with the ALU in arithmetic operations.

- Register B is 12 bits wide and is intended for use with the ALU in arithmetic operations.

- Register C is a 12-bit register which is intended for temporary storage. It has a bidirectional, bus capability.

Memory Address Register -- The memory address register (MAR) is an eight-bit register used for memory data references. The MAR is divided into two sections as shown in the sketch below. The upper four bits, 3-6, are used to indicate a particular block of detector data in the preprocessor memory. The lower three bits are used to indicate particular detector data within a particular block of data. Bit 7 is unused. The entire register can be incremented or decremented by one, in binary sequence, upon command.

| 7 | 6 3 | 2 0 |
|---|---|---|
| N<br>U | | |

Jump Register -- The jump register, which is eight bits wide, is used to maintain a stable jump address during the time when the contents of the control memory are changing.

Control Memory Address Register -- The control memory address register is an eight-bit counter register which provides a sequential, binary address for the control memory. This register is capable of being located (bit parallel) from the jump register, incremented as a binary counter, and cleared to zero.

Control Memory (CM) -- The preprocessor instructions are organized on a 12-bit word basis. The upper four bits, most significant, contain the coded instruction. The lower eight bits contain address and/or data as the instruction requires.

| 11 8 | 7 0 |
|---|---|
| Instr. | Data or Address |

Detector Interface -- The detector interface provides data and control paths between the detectors and the preprocessor. A function diagram is shown in Figure 38.

Data Path -- The interface data path starts with a 16-channel analog multiplexer which selectively transfers each detector signal to the A/D converter. The A/D converter converts the analog signature amplitude output to an eight-bit, bit-parallel data word in 4 μsec. Each detector is sampled every 4 msec. The two digital MUXs enable the preprocessor to monitor the status of each detector.

Instructions -- The preprocessor instructions are defined in the following paragraphs along with their machine codes and mneumonics.

Arithmetic (AR) -- All arithmetic instructions shall be defined by this operational code in addition to specific instructions listed in Table 3:

$$\overset{11 \qquad\quad 8}{\boxed{0001}}$$

In Table 3, the following definitions apply for the symbols contained therein:

A = Contents of Register A1

B = Contents of Register B

F = Output (i.e., "F = A minus B" means output is equal to the result of subtracting the contents of Register B from the contents of Register A)

+ = Logical "OR" (i.e., "A + B" means contents of Register A1 logically "AND'd" with contents of Register B)

$\overline{A}$ = Logical "NOT" (or inversion) of function with the bar over it [i.e., "$\overline{AB}$" means contents of Register A1 logically "AND'd" with the contents of Register B and the resultant function is logically "NOT'd" (inverted)]

All subtracting operation are in 2's complement arithmetic.

Bits 6 and 7 are spare and are to be at logical "0".

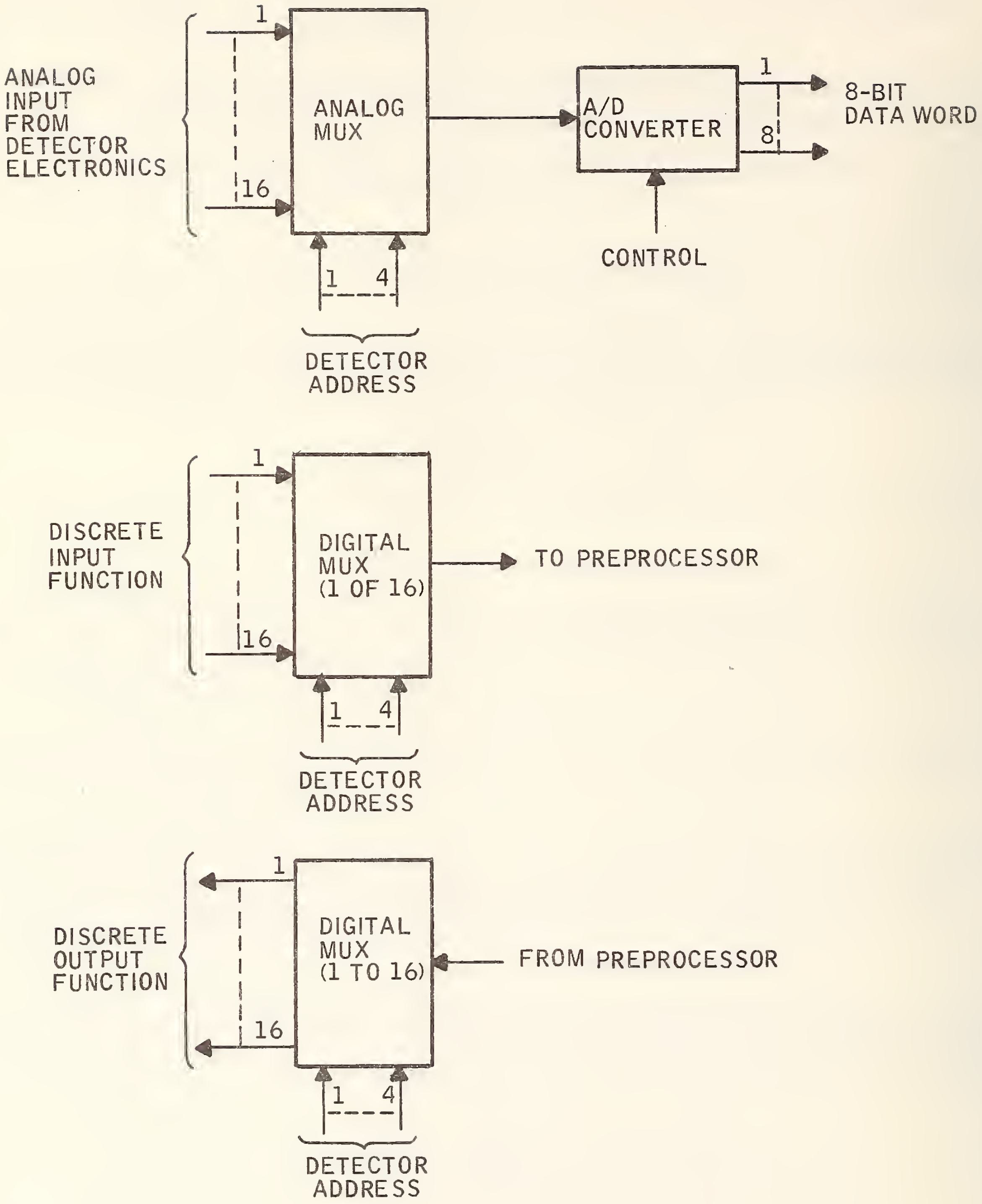


Figure 38. Detector Interface Block Diagram

Table 3. Arithmetic Instructions

| Bits 0-3<br>3210 | Bit 4 = 1<br>Logic<br>Functions | Bit 4 - 0 Arithmetic Operations | |
|---|---|---|---|
| | | Bit 5 = 1 | Bit 5 = 0 |
| 0000 | $F = \overline{A}$ | $F = A$ | $F = A$ plus 1 |
| 0001 | $F = \overline{A + B}$ | $F = A + B$ | $F = (A + B)$ plus 1 |
| 0010 | $F = \bar{A}B$ | $F = A + B$ | $F = (A + \overline{B})$ plus 1 |
| 0011 | $F = 0$ | F = Minus 1 (2's compl) | F = Zero |
| 0100 | $F = \overline{AB}$ | $F = A$ plus $AB$ | $F = A$ plus $A\overline{B}$ plus 1 |
| 0101 | $F = \overline{B}$ | $F = (A + B)$ plus $A\overline{B}$ | $F = (A + B)$ plus $A\overline{B}$ plus 1 |
| 0110 | $F = A \oplus B$ | $F = A$ minus $B$ minus 1 | $F = A$ minus $B$ |
| 0111 | $F = A\overline{B}$ | $F = AB$ minus 1 | $F = A\overline{B}$ |
| 1000 | $F = \overline{A} + B$ | $F = A$ plus $AB$ | $F = A$ plus $AB$ plus 1 |
| 1001 | $F = \overline{A \oplus B}$ | $F = A$ plus $B$ | $F = A$ plus $B$ plus 1 |
| 1010 | $F = B$ | $F = (A + \overline{B})$ plus $AB$ | $F = (A + B)$ plus $A\overline{B}$ plus 1 |
| 1011 | $F = AB$ | $F = AB$ minus 1 | $F = AB$ |
| 1100 | $F = 1$ | $F = A$ plus $A$* | $F = A$ plus $A$ plus 1 |
| 1101 | $F = A + \overline{B}$ | $F = (A + B)$ plus $A$ | $F = (A + B)$ plus $A$ plus 1 |
| 1110 | $F = A + B$ | $F = (A + B)$ plus $A$ | $F = (A + \overline{B})$ plus $A$ plus 1 |
| 1111 | $F = A$ | $F = A$ minus 1 | $F = A$ |

*Each bit is shifted to the next more significant position.

Jump Instructions -- The jump instructions are organized as shown below. Bits 8-11 indicate the operational code, while bits 0-7 contain the jump address. If the jump condition is true, the jump address is loaded into the CMAR.

| 11 8 | 7 0 |
|---|---|
| Op Code | Address |

- Jump if > 0 (JMPG) -- If the contents of the accumulator, $A_0$, is greater than zero, the CMAR is loaded with the jump address.

| 11 8 |
|---|
| 0010 |

- Jump if < 0 (JMPL) -- If the contents of the accumulator, $A_0$, is less than or equal to zero, the CMAR is loaded with the jump address.

| 11 8 |
|---|
| 0011 |

- Jump if = 0 (JMPO) -- If the content of the accumulator, $A_0$, is equal to zero, the CMAR is loaded with the jump address.

| 11 8 |
|---|
| 0100 |

- Jump Unconditional (JMPU) -- The CMAR is loaded with the jump address.

| 11 8 |
|---|
| 0101 |

Transfer Instructions (TR) -- The transfer instructions are organized as shown below:

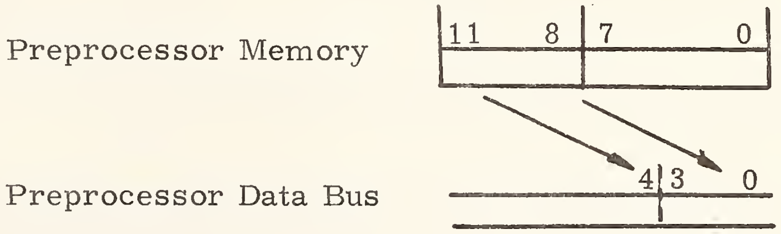
| 11 8 | 7 4 | 3 0 |
|---|---|---|
| 0110 | Source | Dest |

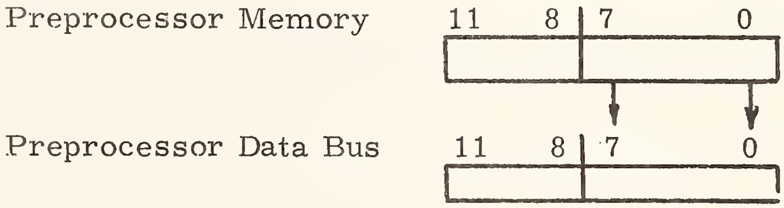
The destination, bits 0-3, indicates where data on the preprocessor data bus go. The source, bits 4-7, shall indicate where the data originate.

Sources -- The following codes cause the appropriate registers, etc., to place data on the preprocessor data bus:

| Code | Source | Mnemonic |
|---|---|---|
| 0001 | Preprocessor Memory, bits 0-11 | NEM |
| 0010 | Preprocessor Memory Lower, bits 0-7 | MEML |
| 0011 | Preprocessor Memory Upper, bits 8-11 | MEMU |
| 0100 | $A_0$ | A0 |
| 0101 | C | C |
| 0110 | Data Input Register | DIR |
| 0111 | T-Switch | T |
| 1000 | E-Switch | E |
| 1001 | Memory Address Register | MAR |

The memory upper and memory lower sources need explanation. The memory upper source code causes memory bits 8-11 to be placed on the preprocessor data bus bits 0-3. The memory lower source code causes memory bits 0-7 to be placed on the preprocessor data bus bits 0-7. Memory bits not directly involved with these operations are not affected:

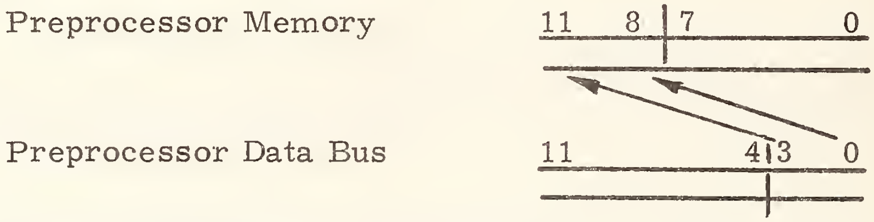


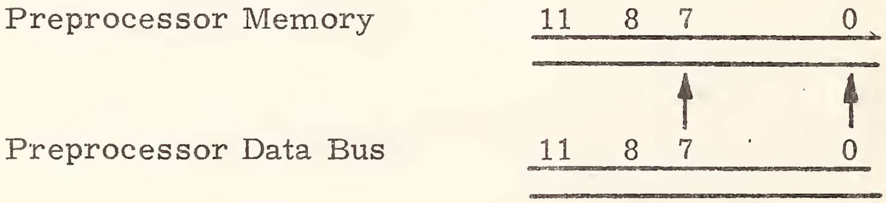


Destinations -- The following codes cause the data appearing on the Preprocessor data bus to be loaded into the appropriate location, register, etc:

| Code | Destination | Mnemonic |
|---|---|---|
| 0001 | Preprocessor Memory, bits 0-11 | NEM |
| 0010 | Preprocessor Memory Lower, bits 0-7 | NEML |
| 0011 | Preprocessor Memory Upper, bits 8-11 | MEMU |
| 0100 | $A_1$ | A1 |
| 0101 | B | B |
| 0110 | C | C |
| 0111 | MAR | MAR |
| 1000 | Main Memory Data Register | MMDR |
| 1001 | Main Memory Address Register | MMAR |
| 1010 | $A_0$ | A0 |

The memory upper and memory lower codes need explanation. The memory upper code causes the preprocessor data bus bits 0-3 to be stored in preprocessor memory bits 8-11. The memory lower code causes the preprocessor data bus bits 0-7 to be stored in preprocessor memory bits 0-7. Memory bits not directly involved with these operations are not affected.





Move Instruction -- This instruction is used where a numerical constant is desired for mathematical operations. Bits 0-7 of the CM will contain the data.

| 11 8 | 7 0 |
|---|---|
| XXXX | Data |

- CM to Register C (CMC) -- The contents of the CM bits 0-7 are transferred to Register C. Bits 8-11 of Register C are forced to zero.

| 11 8 |
|---|
| 0111 |

- CM to Register B (CMB) -- The contents of the CM bits 0-7 are transferred to Register B. Bits 8-11 of Register B are forced to zero.

| 11 8 |
|---|
| 1000 |

Branch Instructions (BOCX) -- These instructions are similar to jump instructions in that bits 0-7 contain the address which, if the branch condition is true, is loaded into the CMAR. Bits 8-11 define the specific branch condition as follows:

| 11 8 | 7 0 |
|---|---|
| XXXX | Address |

| Code | Designation/Mnemonic | Condition |
|---|---|---|
| 1001 | BOC 1 | Processor not initialized |
| 1010 | BOC 2 | Detector not initialized |
| 1011 | BOC 3 | ALU carry |
| 1100 | BOC 4 | Bit 0 of A1 equals zero |

Discrete Output Instructions (ODSC-) -- These instructions provide the capability of pulsing specific control lines to various interfaces. The instruction format is specified as follows. Bits 8-11 designate this class of instructions. Bits 0-3 designate the specific function number as follows:

| 11 8 |
|---|
| 1110 |

| 11 8 | 7 4 | 3 0 |
|---|---|---|
| 1110 | 0 0 | XXX |

| 3 0<br>XXXX | Designation | Function |
|---|---|---|
| 0001 | Output Discrete 1 | Set preprocessor initialization flag |
| 0010 | Output Discrete 2 | Set detector initialization flag |
| 0100 | Output Discrete 3 | Start A/D conversion |
| 1000 | Output Discrete 4 | Spare |
| 0101 | Output Discrete 5 | Spare |
| 0110 | Output Discrete 6 | Store in main memory |

General Instructions (GN) -- The instructions which follow have the following format:

| 11 8 |
|---|
| 1101 |

| 11 8 | 7 4 | 3 0 |
|---|---|---|
| 1101 | XXXX | 0000 |

Bits 8-11 designate this class of instructions. Bits 0-3 designate the specific instruction.

- Increment MAR (IMAR) -- This instruction causes the MAR to advance by one binary county.

| 7 4 |
|---|
| 0001 |

- Decrement MAR (DMAR) -- This instruction causes the MAR to decrement the address by one binary count.

| 7 4 |
|---|
| 0010 |

- Clear B Register (CLRB) -- This instruction causes the contents of the B Register to be cleared to zero.

| 7 4 |
|---|
| 0011 |

- Clear C Register (CLRC) -- This instruction causes the contents of the C Register to be cleared to zero.

| 7 | 4 |
|---|---|
| 0100 | |

- Halt Instruction (HALT) -- This instruction causes the preprocessor to stop the CMAR from being incremented and causes all preprocessor timing to cease.

| 7 | 4 |
|---|---|
| 0101 | |

- No Operation (NOOP) -- When the contents of the control memory are all zeros, no operation is performed.

| 11 | 0 |
|---|---|
| 0 | 0 |

Special Functions --

- Single Instruction -- The preprocessor is implemented with the capability of stepping the CMAR one address at a time. The switch overrides the timing pulse which advances this register and, instead, advances the register one address. A complete fetch-execute cycle takes place each time the Single Instruction switch is depressed.

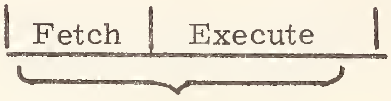
- Interrupt -- An interrupt is generated in the preprocessor timing which has a 250-μsec period. This interrupt is disabled prior to preprocessor initialization and/or when the preprocessor is in a single-instruction mode and/or when the preprocessor is halted. This interrupt sets the detector sampling rate.

- Run Switch -- This switch is used in conjunction with the halt instruction. The switch is of the momentary-contact type. Pushing this switch causes the preprocessor to resume timing where the CMAR was stopped.

- Halt Enable Switch -- This switch is capable of preventing the halt instruction from halting the preprocessor.

- Halt Switch -- This switch, a momentary-contact type, is capable of halting the preprocessor without clearing any registers and is independent of the Halt Enable switch.

- Preprocessor Timing -- The preprocessor, with the exception of the detector interface, operates on a fetch-execute cycle which comprises the basic timing pattern. The fetch portion of the instruction cycle is defined to be when the CMAR is advanced to the next address, and the instruction at that address is read and decoded. The execution portion of the instruction cycle is where the decoded instruction is performed and the result is stable. The preprocessor instruction time is 2 μsec maximum for any and all instructions.



One Instruction Time (Cycle)

Program -- The program listing for the preprocessor is presented in Table 4. Column X of the binary code contains the most significant bits.

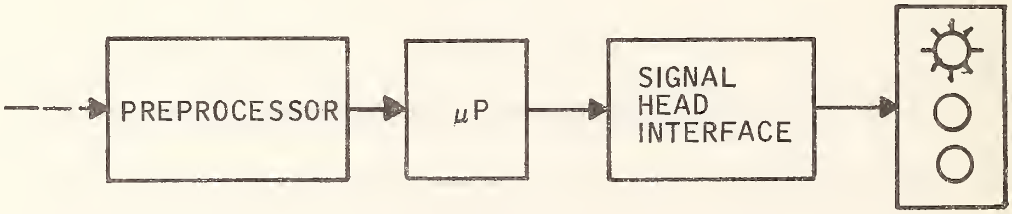
Hardware Description -- Table 5 identifies the major components of the preprocessor schematic, sheet-by-sheet, excluding the timing diagram.

Signal Names -- Table 6 identifies those signal names most significant to understanding signal flow.

Figure 39 explains the switches, etc., on top of Card 1 of the preprocessor. Serial No. 1 Preprocessor is in the Minneapolis BPCU at this time.

## Micrcprocessor (μP)

The microprocessor has the dual function of performing the vehicle classification algorithm and executing signal head controller functions. The following sketch shows the functional relationship between these two processes.



## Table 4. Bus Detector Preprocessor Program Listing

INCMAR = IMAR, MEMU = MEMUPR

| DEC ADX | | | BIN ADX | X Y Z | |
|---|---|---|---|---|---|
| 000 | NOOP | | 000 | 000000000000 | 000 |
| 001 | BOC 1, $243_{10}$ | | 1 | 100111110011 | 9F3 |
| 002 | ODSC, 3 | | 2 | 111000000011 | E03 |
| 003 | BOC 2, $193_{10}$ | To detector initialization | 3 | 101011000001 | AC1 |
| 004 | NOOP | | 4 | | 000 |
| 005 | NOOP | A/D Timer | 5 | | 000 |
| 006 | NOOP | | 6 | | 000 |
| 007 | TR, MEM, B | | 7 | 011000010101 | 615 |
| 008 | TR, DIR, A1 | Is vehicle present? | 010 | 011001100100 | 664 |
| 009 | AR, SUB | $(K_n - b_i)$ | 1 | 000100000110 | 106 |
| 010 | TR, AO, C | | 2 | 011001000110 | 646 |
| 011 | JMPL, 105 | | 3 | 001101101001 | 369 |
| 012 | TR, A0, A1 | | 4 | 011001000100 | 644 |
| 013 | TR, T, B | | 5 | 011001110101 | 675 |
| 014 | AR, SUB | | 6 | 000100000110 | 106 |
| 015 | JMPL, $135_{10}$ | | 7 | 001110000111 | 387 |
| 016 | GN, IMAR | | 020 | 110100010000 | D10 |
| 017 | TR, MEMU, A1 | | 1 | 011000110100 | 634 |
| 018 | BOC4, $039_{10}$ | Presence flag set? | 2 | 110000100111 | C27 |
| 019 | AR, A1 + 1 | | 3 | 000100000000 | 100 |
| 020 | TR, AO, MEMU | Set presence flag | 4 | 011001000011 | 643 |
| 021 | GN, IMAR | | 5 | 110100010000 | D10 |
| 022 | TR, AO, MEM | Inc Int. Timer | 6 | 011001000001 | 641 |
| 023 | GN, IMAR | | 7 | 110100010000 | D10 |
| 024 | TR, AO, MEMU | Set positive slope flag | 030 | 011001000011 | 643 |
| 025 | TR, MAR, A1 | | 1 | 011010010100 | 694 |
| 026 | CMB, 2 | | 2 | 100000000010 | 802 |
| 027 | AR, ADD | | 3 | 000100101001 | 129 |
| 028 | TR, AO, MMAR | | 4 | 011001001001 | 649 |
| 029 | TR, A0, A1 | | 5 | 011001000100 | 644 |
| 030 | TR, C, AO | | 6 | 011001011010 | 65A |
| 031 | GN, CLRC | | 7 | 110101000000 | D40 |
| 032 | TR, C, MMDR | | 040 | 011001011000 | 658 |
| | | DECMAR = DMAR | | | |
| 033 | ODSC, 6 | Clr last ampl. data | 041 | 111000000110 | E06 |
| 034 | TR, AO, C | | 2 | 011001000110 | 646 |
| 035 | AR, ADD | | 3 | 000100101001 | 129 |
| 036 | TR, AO, MMAR | | 4 | 011001001001 | 649 |
| 037 | ODSC, 6 | Clr status flag | 5 | 111000000110 | E06 |
| 038 | JMPU, $045_{10}$ | | 6 | 010100101101 | 52D |
| 039 | GN, IMAR | | 7 | 110100010000 | D10 |

## Table 4. Bus Detector Preprocessor Program Listing (Continued)

DECMAR = DMAR

| DEC ADX | | | BIN ADX | X Y Z | |
|---|---|---|---|---|---|
| 040 | TR, MEM, A1 | | 050 | 011000010100 | 614 |
| 041 | AR, A1+1 | | 1 | 000100000000 | 100 |
| 042 | BOC3, 044 | Inc Int. Timer | 2 | 101100101100 | 82C |
| 043 | TR, AO, MEM | | 3 | 011001000001 | 641 |
| 044 | GN, IMAR | | 4 | 110100010000 | D10 |
| 045 | GN, IMAR | | 5 | 110100010000 | D10 |
| 046 | TR, C, A1 | MAR at $K_{n-1}$ | 6 | 011001010100 | 654 |
| 047 | TR, MEM, 8 | | 7 | 011000010101 | 615 |
| 048 | AR, SUB | Which slope? | 060 | 000100000110 | 106 |
| 049 | JMPO, $108_{10}$ | | 1 | 010001101100 | 46C |
| 050 | JMPG, $108_{10}$ | | 2 | 001001101100 | 26C |
| 051 | GN, DMAR | | 3 | 110100100000 | D20 |
| 052 | TR, MEMU, A1 | | 4 | 011000110100 | 634 |
| 053 | BOC4, $055_{10}$ | Slope flag set? | 5 | 110000110111 | C37 |
| 054 | JMPU, $124_{10}$ | | 6 | 010101111100 | 57C |
| 055 | GN, IMAR | | 7 | 110100010000 | D10 |
| 056 | TR, MEM, A1 | | 070 | 011000010100 | 614 |
| 057 | TR, C, B | | 1 | 011001010101 | 655 |
| 058 | AR, SUB | | 2 | 000100000110 | 106 |
| 059 | JMPL, $126_{10}$ | Tolerance > ε ? | 3 | 001101111110 | 37E |
| 060 | TR, A0, A1 | | 4 | 011001000100 | 644 |
| 061 | TR, ε, B | | 5 | 011010000101 | 685 |
| 062 | AR, SUB | | 6 | 000100000110 | 106 |
| 063 | JMPL, $126_{10}$ | | 7 | 001101111110 | 37E |
| 064 | TR, MEM, MMDR | Save peak value | 100 | 011000011000 | 618 |
| | | MEMLWR = MEML | | | |
| 065 | TR, C, MEM | Store $k_n$-$b_i$ in $k_{n-1}$ | 101 | 011001010001 | 651 |
| 066 | GN, IMAR | | 2 | 110100010000 | D10 |
| 067 | TR, MEMU, A1 | | 3 | 011000110100 | 634 |
| 068 | CMB, 3 | | 4 | 100000000011 | 803 |
| 069 | AR, SUB | Inc # peaks if < 3 | 5 | 000100000110 | 106 |
| 070 | JMPO, $078_{10}$ | | 6 | 010001001110 | 44E |
| 071 | JMPG, $078_{10}$ | | 7 | 001001001110 | 24E |
| 072 | AR, A1+1 | | 110 | 000100000000 | 100 |
| 073 | TR, AO, MEMU | | 1 | 011001000011 | 643 |
| 074 | TR, MEML, A1 | | 2 | 011000100100 | 624 |
| 075 | AR, A1+1 | Inc ADX index | 3 | 000100000000 | 100 |
| 076 | TR, AO, MEML | | 4 | 011001000010 | 642 |
| 077 | JMPU, $081_{10}$ | | 5 | 010101010001 | 551 |
| 078 | TR, MEML, 41 | | 6 | 011000100100 | 624 |
| 079 | AR, A1-1 | Dec ADX index | 7 | 000100101111 | 12F |
| 080 | TR, AO, MEML | | 120 | 011001000010 | 642 |

Table 4. Bus Detector Preprocessor Program Listing (Continued)

MEMLWR = MEML

| DEC ADX | | | BIN ADX | X Y Z | |
|---|---|---|---|---|---|
| 081 | TR, MAR, A1 | | 121 | 011010010100 | 694 |
| 082 | CMB, $370_8$ | | 2 | 100011111000 | 8F8 |
| 083 | AR, AND | | 3 | 000100011011 | 118 |
| 084 | TR, MEML, B | Get ADX in store peak | 4 | 011000100101 | 625 |
| 085 | TR, A0, A1 | | 5 | 011001000100 | 644 |
| 086 | AR, ADD | | 6 | 000100101001 | 129 |
| 087 | TR, AO, MMAR | | 7 | 011001001001 | 649 |
| 088 | ODSC, 6 | | 130 | 111000000110 | E06 |
| 089 | TR, A0, A1 | | 1 | 011001000100 | 644 |
| 090 | AR, A1+1 | | 2 | 000100000000 | 100 |
| 091 | TR, AO, MEML | MAR at ADX index | 3 | 011001000010 | 642 |
| 092 | TR, AO, C | | 4 | 011001000110 | 646 |
| 093 | GN, DMAR | | 5 | 110100100000 | D20 |
| 094 | GN, DMAR | Update ADX index and store time interval | 6 | 110100100000 | D20 |
| 095 | GN, DMAR | | 7 | 110100100000 | D20 |
| 096 | TR, MEM, MMDR | | 140 | 011000011000 | 618 |
| 097 | TR, C, MMAR | | 1 | 011001011001 | 659 |
| 098 | ODSC, 6 | | 2 | 111000000110 | E06 |
| 099 | TR, MAR, A1 | | 3 | 011010010100 | 694 |
| 100 | CMB, 6 | Update next detector address | 4 | 100000000110 | B06 |
| 101 | AR, ADD | | 5 | 000100101001 | 129 |
| 102 | TR, AO, MAR | | 6 | 011001000111 | 647 |
| 103 | NOOP | | 7 | 000000000000 | 000 |
| 104 | JMPU, $103_{10}$ | | 150 | 010101100111 | 567 |
| 105 | GN, CLRC | | 1 | 110101000000 | D40 |
| 106 | JMPU, 135 | | 2 | 010110000111 | 587 |
| 107 | GN, HALT | Should not be executed.<br>MAR at $k_{n-1}$ | 3 | 110101010000 | D50 |
| 108 | GN, DEC MAR | | 4 | 110100100000 | D20 |
| 109 | TR, MEMUPR, A1 | | 5 | 011000110100 | 634 |
| 110 | BOC4, 124 | Slope flag set? | 6 | 110001111100 | C7C |
| 111 | GN, INCMAR | | 7 | 110100010000 | D10 |
| 112 | TR, C, A1 | | 160 | 011001010100 | 654 |
| 113 | TR, MEM, B | | 1 | 011000010101 | 615 |
| 114 | AR, SUB | | 2 | 000100000110 | 106 |
| 115 | TR, A0, A1 | | 3 | 011001000100 | 644 |
| 116 | TR, ε, B | | 4 | 011010000101 | 685 |
| 117 | AR, SUB | | 5 | 000100000110 | 106 |
| 118 | JMPL, 126 | | 6 | 001101111110 | 37E |
| 119 | ODSC, 4 | For debug purposes only. | 7 | 111000000100 | E04 |
| 120 | GN, DECMAR | | 170 | 110100100000 | D20 |
| 121 | TR, MEMUPR, A1 | | 1 | 011000110100 | 634 |
| 122 | AR, A1+1 | Set slope flag | 2 | 000100000000 | 100 |
| 123 | TR, AO, MEMUPR | | 3 | 011001000011 | 643 |

## Table 4. Bus Detector Preprocessor Program Listing (Continued)

| DEC ADX | | MEMLWR = MEML | BIN ADX | X Y Z | |
|---|---|---|---|---|---|
| 124 | GN, INCMAR | | 174 | 110100010000 | D10 |
| 125 | TR, C, MEM | Store $(k_n-b_i)$ in $k_{n-1}$<br>CM = 11111000 | 5 | 011001010001 | 651 |
| 126 | TR, MAR, A1 | Update next detector address. CM = 8 | 6 | 011010010100 | 694 |
| 127 | CMB, $370_8$ | | 7 | 100011111000 | 8F8 |
| 128 | AR, AND | | 200 | 000100011011 | 11B |
| 129 | TR, A0, A1 | | 1 | 011001000100 | 644 |
| 130 | CMB, 8 | | 2 | 100000001000 | 808 |
| 131 | AR, ADD | | 3 | 000100101001 | 129 |
| 132 | TR, AO, MAR | | 4 | 011001000111 | 647 |
| 133 | NOOP | | 5 | 000000000000 | 000 |
| 134 | JMPU, 133 | | 6 | 010110000101 | 585 |
| 135 | GN, INCMAR | Presence flag set? | 7 | 110100010000 | D10 |
| 136 | TR, MEMUPR. A1 | | 210 | 011000110100 | 634 |
| 137 | BOC4, 139 | | 1 | 110010001011 | C88 |
| 138 | JMPU, 192 | To long term signal average | 2 | 010111000000 | 5CO |
| 139 | TR, C, A1 | Hysteresis check | 3 | 011001010100 | 654 |
| 140 | AR, A1+1 | | 4 | 000100000000 | 100 |
| 141 | TR, A0, A1 | | 5 | 011001000100 | 644 |
| 142 | TR, T, B | | 6 | 011001110101 | 675 |
| 143 | AR, SUB | | 7 | 000100000110 | 106 |
| 144 | JMPG. 175 | | 220 | 001010101111 | 2AF |
| 145 | TR, MEMUPR, A1 | Reset presence flag | 1 | 011000110100 | 634 |
| 146 | AR, A1-1 | | 2 | 000100101111 | 12F |
| 147 | TR, AO, MEMUPR | | 3 | 011001000011 | 643 |
| 148 | GN, DECMAR | | 4 | 110100100000 | D20 |
| 149 | TR, MAR, C | MAR at $b_i$ | 5 | 011010010110 | 696 |
| 150 | GN, INCMAR | | 6 | 110100010000 | D10 |
| 151 | GN, INCMAR | | 7 | 110100010000 | D10 |
| 152 | TR, MEM, A1 | Update and store last interval | 230 | 011000010100 | 614 |
| 153 | AR, A1+1 | | 1 | 000100000000 | 100 |
| 154 | BOC3, 157 | | 2 | 101110011101 | B9D |
| 155 | TR, AO, MMDR | | 3 | 011001001000 | 648 |
| 156 | JMPU, 158 | | 4 | 010110011110 | 59E |
| 157 | TR, MEM, MMDR | | 5 | 011000011000 | 618 |
| 158 | TR, C, MMAR | | 6 | 011001011001 | 659 |
| 159 | ODSC6 | | 7 | 111000000110 | E06 |
| 160 | TR, C, A1 | | 240 | 011001010100 | 654 |
| 161 | GN, CLRC | | 1 | 110101000000 | D40 |
| 162 | TR, C, MEM | Clr interval | 2 | 011001010001 | 651 |

## Table 4. Bus Detector Preprocessor Program Listing (Continued)

| DEC ADX | | MEMLWR = MEML | BIN ADX | X Y Z | |
|---|---|---|---|---|---|
| 163 | CMB, 7 | | 243 | 100000000111 | 807 |
| 164 | AR, ADD | | 4 | 000100101001 | 129 |
| 165 | TR, AO, MMAR | Status adx | 5 | 011001001001 | 649 |
| 166 | GN, INCMAR | | 6 | 110100010000 | D10 |
| 167 | GN, INCMAR | | 7 | 110100010000 | D10 |
| 168 | TR, C, MEM | Update adx and store interval and clear flags | 250 | 011001010001 | 651 |
| 169 | GN, INCMAR | | 1 | 110100010000 | D10 |
| 170 | TR, C, MEM | | 2 | 011001010001 | 651 |
| 171 | CMC, 1 | CM = 1 | 3 | 011100000001 | 701 |
| 172 | TR, C, MMDR | Set status bit | 4 | 011001011000 | 658 |
| 173 | ODSC6 | | 5 | 111000000110 | E06 |
| 174 | JMPU, 126 | | 6 | 010101111110 | 57E |
| 175 | GN, INCMAR | | 7 | 110100010000 | D10 |
| 176 | TR, MEM, A1 | | 260 | 011000010100 | 614 |
| 177 | AR, A1+1 | Increment interval | 1 | 000100000000 | 100 |
| 178 | BOC3, 180 | | 2 | 101110110100 | BB4 |
| 179 | TR, AO, MEM | | 3 | 011001000001 | 641 |
| 180 | JMPU, 126 | | 4 | 010101111110 | 57E |
| 181 | | | 5 | | |
| 182 | | | 6 | | |
| 183 | | | 7 | | |
| 184 | | | 270 | | |
| 185 | | | 1 | | |
| 186 | | | 2 | | |
| 187 | GN, CLRC | | 3 | 110101000000 | D40 |
| 188 | TR, C, MAR | This loop for debug only | 4 | 011001010111 | 657 |
| 189 | GN, IMAR | | 5 | 110100010000 | D10 |
| 190 | JMPU, 189 | | 6 | 010110111101 | 5BD |
| 191 | GN, HALT | Program should not get here | 7 | 110101010000 | D50 |
| 192 | GN, DMAR | Entry from FEX | 300 | 110100100000 | D20 |
| 193 | TR, MAR, C | Entry from BOC2; save MAR | 1 | 011010010110 | 696 |
| 194 | TR, MAR, A1 | | 2 | 011010010100 | 694 |
| 195 | CMB, 6 | | 3 | 100000000110 | 806 |
| 196 | AR, ADD | | 4 | 000100101001 | 129 |
| 197 | TR, AO, MAR | | 5 | 011001000111 | 647 |
| 198 | TR, MEM, A1 | | 6 | 011000010100 | 614 |
| 199 | BOC2, 232 | | 7 | 101011101000 | AE8 |
| 200 | CMB, $256_{10}$ | | 310 | 100011111111 | 8FF |
| 201 | AR, SUB | | 1 | 000100000110 | 106 |
| 202 | JMPL, 209 | | 2 | 001111010001 | 3D1 |
| 203 | TR, A0, A1 | | 3 | 011001000100 | 644 |
| 204 | CMB, $119_{10}$ | Long-term signal average and detector initialization | 4 | 100001110111 | 877 |
| 205 | AR, SUB | | 5 | 000100000110 | 106 |

Table 4. Bus Detector Preprocessor Program Listing (Continued)

| DEC ADX | | MEMLWR = MEML | BIN ADX | X Y Z | |
|---|---|---|---|---|---|
| 206 | JMPL, 208 | | 316 | 001111010000 | 3DO |
| 207 | JMPO, 216 | | 7 | 010011011000 | 4D8 |
| 208 | TR, MEM, A1 | | 320 | 011000010100 | 614 |
| 209 | AR, A1+1 | | 1 | 000100000000 | 100 |
| 210 | TR, AO, MEM | | 2 | 011001000001 | 641 |
| 211 | TR, MAR, A1 | | 3 | 011010010100 | 694 |
| 212 | CMB, 2 | | 4 | 100000000010 | 802 |
| 213 | AR, ADD | | 5 | 000100101001 | 129 |
| 214 | TR, AO, MAR | | 6 | 011001000111 | 647 |
| 215 | JMPU, 240 | | 7 | 010111110000 | 5FO |
| 216 | TR, AO, MEM | Clear sample count | 330 | 011001000001 | 641 |
| 217 | TR, C, MAR | Restore MAR | 1 | 011001010111 | 657 |
| 218 | TR, DIR, B | | 2 | 011001100101 | 665 |
| 219 | TR, MEM, A1 | | 3 | 011000010100 | 614 |
| 220 | AR, SUB | $b_i$-$k_n$ | 4 | 001000000110 | 106 |
| 221 | JMPO, 236 | | 5 | 010011101100 | 4EC |
| 222 | JMPL, 225 | | 6 | 001111100001 | 3E1 |
| 223 | AR, A1-1 | | 7 | 000100101111 | 12F |
| 224 | JMPU, 226 | | 340 | 010111100010 | 5E2 |
| 225 | AR, A1+1 | | 1 | 000100000000 | 100 |
| 226 | TR, AO, MEM | Store new $b_i$ | 2 | 011001000001 | 641 |
| 227 | TR, C, A1 | | 3 | 011001010100 | 654 |
| 228 | CMB, 8 | | 4 | 100000001000 | 808 |
| 229 | AR, ADD | Update new detector address | 5 | 000100101001 | 129 |
| 230 | TR, AO, MAR | | 6 | 011001000111 | 647 |
| 231 | JMPU, 240 | | 7 | 010111110000 | 5FO |
| 232 | CMB, $126_{10}$ | | 350 | 100001111110 | 87E |
| 233 | AR, SUB | | 1 | 000100000110 | 106 |
| 234 | JMPO, 216 | | 2 | 010011011000 | 4DB |
| 235 | JMPU, 209 | | 3 | 010111010001 | 5D1 |
| 236 | BOC2, 238 | | 4 | 101011101110 | AEE |
| 237 | JMPU, 227 | | 5 | 010111100011 | 5E3 |
| 238 | ODSC, 2 | | 6 | 111000000010 | E02 |
| 239 | JMPU, 227 | | 7 | 010111100011 | 5E3 |
| 240 | NOOP | | 360 | 000000000000 | 000 |
| 241 | JMPU, 240 | | 1 | 010111110000 | 5F0 |

## Table 4. Bus Detector Preprocessor Program Listing (Concluded)

MEMLWR = MEML

| DEC ADX | | | BIN ADX | X Y Z | |
|---|---|---|---|---|---|
| 242 | GN, HALT | Program should never reach here | 362 | 110101010000 | D50 |
| 243 | GN, CLRC | pP Initialization | 3 | 110101000000 | D40 |
| 244 | TR, C, MAR | | 4 | 011001010111 | 657 |
| 245 | TR, C, A1 | | 5 | 011001010100 | 654 |
| 246 | TR, C, MEM | | 6 | 011001010001 | 651 |
| 247 | AR, A1+1 | | 7 | 000100000000 | 100 |
| 248 | BOC3, 252 | | 370 | 101111111100 | BFC |
| 249 | TR, A0, A1 | | 1 | 011001000100 | 644 |
| 250 | GN, IMAR | | 2 | 110100010000 | D10 |
| 251 | JMPU, 246 | | 3 | 010111110110 | 5F6 |
| 252 | ODSC, 1 | | 4 | 111000000001 | E01 |
| 253 | TR, C, MAR | | 5 | 011001010111 | 657 |
| 254 | NOOP | | 6 | 000000000000 | 000 |
| 255 | JMPU, 254 | | 7 | 010111111110 | 5FE |

Table 5. Major Components of Preprocessor Schematic (Timing Diagram Excluded)

| Sheet No. | IC Location | Description |
|---|---|---|
| 1 | 3G, 3F | Register A1 |
| | 4F, 4G | Register B |
| 2 | 1D-E, 1E-F, 2D-E | ALU, 12 bits in 4-bit slices |
| | 2A | ALU Overflow FF, A>B, A=B, and A<B FFs |
| 3 | 1G, 2G, 2F | Accumulator, AO |
| 4 | 5R, 5P | Clock Divider |
| 5 | 4K, 5K, 5L | 250-μsec Interrupt Generator |
| 6 | 6N | Switch Debounce FFs |
| | 6L | Halt and Single-Cycle Timing FFs |
| | 6P | Halt and Single-Cycle Ciming FFs |
| 7 | 5B, 6C | Control Memory Address Register/Counter |
| | 4D, 5C | Instruction Register |
| | 5E, 4E, 5D | Control Memory |
| | 6D-E, 2C-D, 6E-F, 4A, 5A | Instruction Decoders |
| | 6A | Preprocessor Initialization FF and Jump/Branch FF |
| | 3A, 2H, 3B, 4C | Jump and Branch Multiplexer |
| | 3H, 4H | Register C |
| | 5F, 5G, 5H, 6G, 6H | Preprocessor Data Bus Drivers |
| 8 | 3G | Memory Input Data Multiplexer |
| | 5K, 3K | Memory Enable Control (write) |
| | 4H, 3H, 1J, 5J, 2F, 6J, 3F, 2J, 4F, 4G, 2G, 1G | RAM Memory |
| | 3J, 4J | Memory Address Register |
| | 1H, 2H, 6G, 5G, 6H, 5H | Preprocessor Data Bus Drivers |
| 9 | 6B-C | 16-Channel Analog Multiplexer |
| | 4A-B-C, 5A-B-C | A/D Converter |
| | 5E, 6E | Preprocessor Data Bus Drivers |
| | 6N, 6P | Detector Status Multiplexers (input) |
| | 6A-B | Detector Output Command Multiplexer |

Table 6. Signal Flow Signal Names

| Signal | Function | Sheets |
|---|---|---|
| BPPD0-11 | Preprocessor Data Bus | 1,3,7,8,9 |
| A00-11 | ALU Data Output | 2,3 |
| FPPINIT | Preprocessor Interrupt Enable | 5, |
| GINTR* | Interrupt | 5 |
| INIT* | Initialization | 5 |
| FCYC | Instruction Cycle Enable | 6 |
| MCM0-11 | Control Memory Data Bus | 7 |
| RCMAR0-7 | Control Memory Address Register Bus | 7 |
| /GBOCJMP | Branch ON Condition and Jump Enable | 7 |
| GSTRTCV | A/D Converter Start Convert Pulse | 7,9 |
| RMAR0-7 | RAM Memory Address Register | 8,9 |
| /GWEN1 | Write Enable, Lower 8 bits | 8 |
| /GWEN2 | Write Enable, Upper 4 bits | 8 |
| /MPP0-11 | Memory Data Output | 8 |
| ADET1-16 | Analog Signal Input | 9 |
| FSTAT1-16 | Loop Detector Status Input | 9 |
| /GINIT1-16 | Loop Detector Initialization Output | 9 |
| AD0-7 | A/D Converter Data Output | 9 |

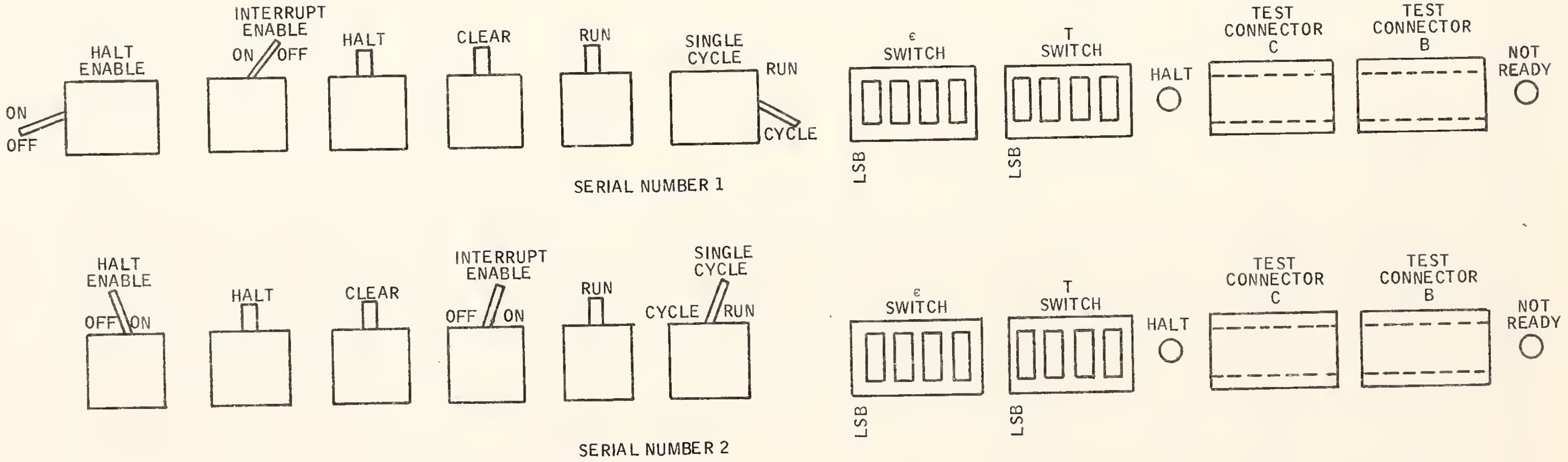


Figure 39. Preprocessor Switch Functions

Vehicle classification is performed based upon the data received from the preprocessor. Simultaneously, the microprocessor cycles through the timing plan, which is contained in memory, and performs functions such as bus preemption, communicating with the loop master, fault monitoring, inputting data and outputting signal information. The signal head interface receives the digital traffic control words from the microprocessor and drives the appropriate signal lights.

One of the first tasks of the bus detector program was to select a suitable microprocessor. This task was somewhat hampered by the fact that it was difficult to know exactly what capabilities were needed since the hardware and software designs were not yet completed. Nevertheless, a decision had to be made. To date, the decision has proved to be a correct one.

The features and capabilities of the microprocessor which were considered in the selection of a particular vendor were the following:

- Availability as a microcomputer system
- Interrupt capability
- Capability of the arithmetic instruction set
- Addressable memory
- Memory word size
- System software
- Programming language

Availability as a Microcomputer System -- The microprocessor had to be available as a microcomputer system to which the required memories and peripheral devices could be obtained and interfaced.

Interrupt Capability -- The MCU requires interrupt capability for timing the real-time control sequence and for computer control unit functions (accessing memory and register contents, for example).

Arithmetic Instruction Set -- The vehicle classification functions present the primary requirements for arithmetic operations in the MCU. The necessary arithmetic operations may be implemented as firmware, microprogrammed instructions or as software subroutines depending on the microprocessor and word length. The required precision may be achieved as double-precision instructions with one microprocessor and achieved as subroutines with another.

The type of arithmetic operations has impact on the precision. The arithmetic may be integer, fixed point or scaled integer, or floating point.

Addressable Memory -- The addressable memory capabilities are evidenced as maximum memory size and addressing modes of the instruction set. Directly addressable memory, the extent of program counter relative addressing, indirect and indexed addressing capabilities are considered.

Memory Word Size -- The memory word size impacts such factors as precision of the arithmetic instruction, parallel processing of discrete information, especially in the area of Input/Output data rates, instruction format length (single, double, or triple words), and addressing capabilities.

System Software -- The system software available is of primary importance to the software development tasks. The system software factors include the compiler or assembler completeness in the areas of directives or pseudo operands, diagnostics, cross reference, and MACRO processing, software required due to the computer environment in which the compilations or assemblies are made, and the interface to the prototyping system's resident software on the prototyping system, and system utilities (loaders, debug programs, self-diagnostics, prom programming, test editing, peripheral I/O drivers, file manipulation).

Programming Language -- The programming language affects the efficiency of the programs and the memory allocation in terms of memory size and type (ROM or RAM).

Vendors -- The choice of microprocessors was quickly narrowed to two systems; the Intellec 8 by Intel, and the IMP-16 by National Semiconductor. A representative of each company was invited to Honeywell to present their systems, answer questions, and provide product literature for our study.

Microprocessor Selection -- Programming flexibility, software development considerations, computation speed, and previous Honeywell experience were the principal factors leading to selection of the National IMP-16P/308 for system development. This microcomputer system is specifically designed for prototype system development. Significant features of this machine are:

- 16-bit word
- 8K read/write memory
- Extended instruction set CROM
- Hardware multiply/divide
- 1.4-μsec microinstruction time

The IMP-16 Applications Manual is presented in Appendix IV. It contains a complete description of the microprocessor's software and hardware capabilities.

Status Display and Monitor Unit (SDMU)

The SDMU is a device which provides the electronic interface between the microprocessor and the signal head circuits. Basically, the SDMU is treated as a peripheral device, similar to TTYs, etc., by the microprocessor. All traffic control functions such as loop master interconnect data, etc., pass through here, which makes the SDMU a sort of central concentrator for all functions downstream for the microprocessor. Commands from the microprocessor are decoded by the SDMU for subsequent action. Two sets of registers, one for main street data and one for cross-street data, provide temporary storage of the traffic light pattern. These registers are also provided with the capability of being read back into the microprocessor for error detection.

On command from the microprocessor, the SDMU performs the flashing Don't Walk function as well as reading bulb and load switch status.

The real-time clock is a crystal-controlled timer which establishes a 0.10-second ± 0.001 percent accuracy for the timing plan.

The watch-dog timer is used to detect a microprocessor stall condition. If the watch-dog timer is not reset within a specified time by the microprocessor, it will automatically initiate a flashing main street red condition.

Subsystem Functional Block Description -- Figure 40 shows the relative physical separation and interconnection between the various subsystem components and subsystem functions.

The Bulb and Load Switch Monitor Unit (BLSMU, a second-generation device, is a separately located electronic box. Control and data functions required by the BLSMU are provided by the BPCU software via the Signal Light Driver and Monitor Unit (SDMU).

The Conflict Monitor Unit (CMU) is a separately located electronic box. Access to the CMU by the BPCU computer is obtained via the SDMU.

The Light Driver Unit (LDU) consists of solid-state relays which perform 60-Hz power switching for the signal lights.

All 60-Hz power/monitor circuits for the signal lights are physically external to the Microcomputer Unit (MCU).

Detailed Functional Description --

Watch-dog Timer -- The SDMU contains a timer used by the system to detect a microprocessor stall condition. The output of this time is OR'd with the stop BPCU control command from the BPCU and goes to the Transfer

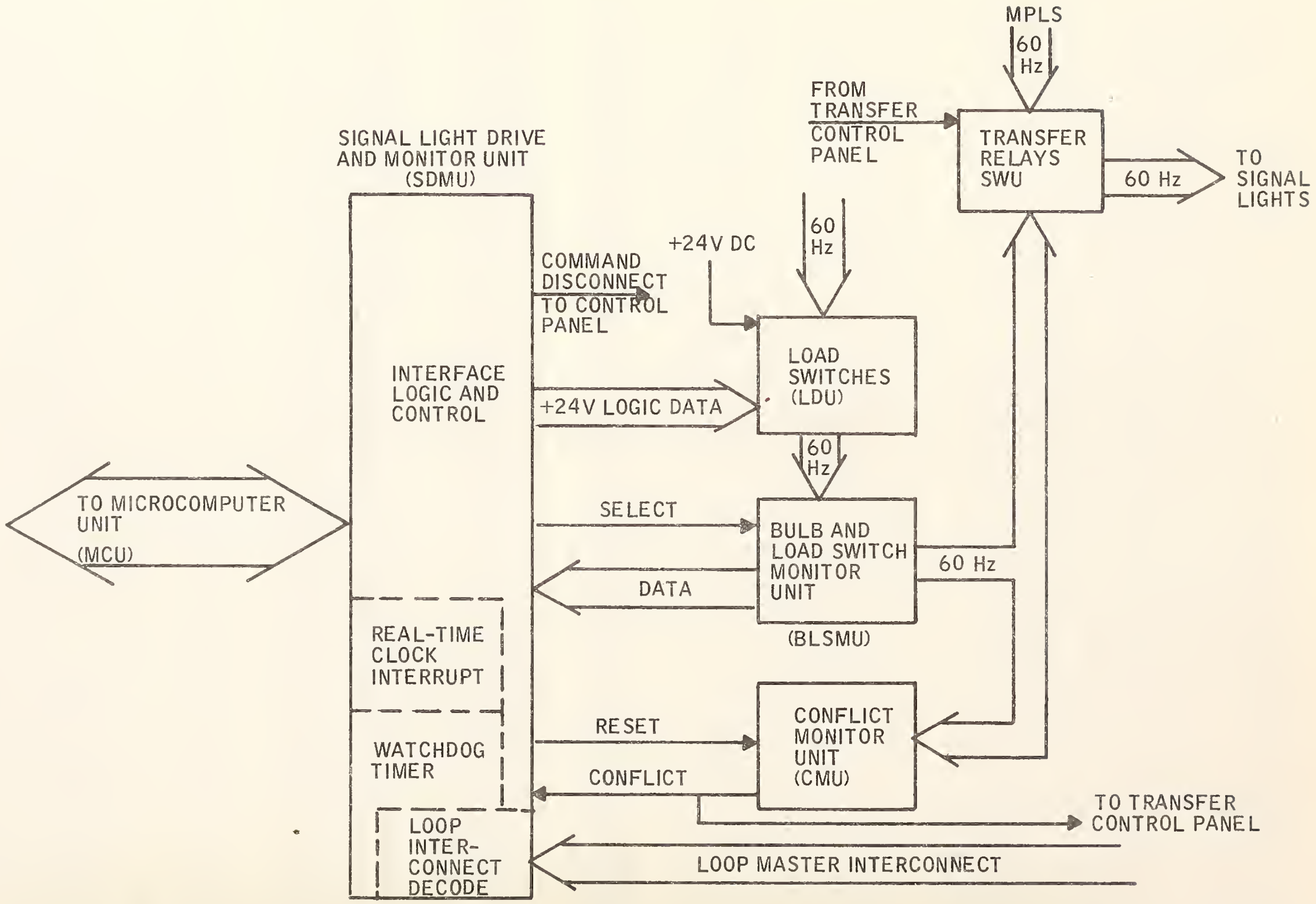

MPLS
60 Hz
FROM TRANSFER CONTROL PANEL
TRANSFER RELAYS SWU
60 Hz
TO SIGNAL LIGHTS
SIGNAL LIGHT DRIVE AND MONITOR UNIT (SDMU)
+24V DC
60 Hz
COMMAND DISCONNECT TO CONTROL PANEL
LOAD SWITCHES (LDU)
INTERFACE LOGIC AND CONTROL
+24V LOGIC DATA
60 Hz
TO MICROCOMPUTER UNIT (MCU)
SELECT
BULB AND LOAD SWITCH MONITOR UNIT
(BLSMU)
60 Hz
DATA
REAL-TIME CLOCK INTERRUPT
RESET
CONFLICT MONITOR UNIT (CMU)
WATCHDOG TIMER
CONFLICT
TO TRANSFER CONTROL PANEL
LOOP INTER-CONNECT DECODE
LOOP MASTER INTERCONNECT

Figure 40. SDMU Interface Diagram

Control Panel (TCP) where it causes the transfer relays to disconnect the BPCU from the signals. This timer is resettable by a software command to the SDMU. The time from the last reset command to timer output is adjustable with a screwdriver (not accessible from outside the BPCU chassis) over a range of 0. 1 second to 0. 5 second. The timer output remains in a Logic One state until reset by software command. The control signal to the TCP is active in the high state (Logic "1") such that if there is a power failure in the SDMU, the TCP disconnects the BPCU. When power is first applied to the SDMU, the timer is inhibited, via system clear, from disconnecting the BPCU from the signals until completion of MCU software initialization. At the end of initialization, the software issues a start command which enables the watch-dog timer to begin its function.

Bulb and Load Switch Monitor Unit (BLSMU) -- In the second generation, the SDMU communicates with the BLSMU by 22 data lines. The data lines carry bulb monitor data (main and cross) from the BLSMU to the SDMU.

| Line Count | | |
|---|---|---|
| Main | Cross | Function |
| 1 | 12 | Red primary flow |
| 2 | 13 | Yellow primary flow |
| 3 | 14 | Green primary flow |
| 4 | 15 | Red secondary flow |
| 5 | 16 | Yellow secondary flow |
| 6 | 17 | Green secondary flow |
| 7 | 18 | Don't walk |
| 8 | 18 | Walk |
| 9 | 20 | Special red |
| 10 | 21 | Special yellow |
| 11 | 22 | Special green |

Loop Master Interconnect Decode -- The SDMU receives five buffered input lines from the loop master controller, $D_2$, $D_3$, $R_1$, $R_2$, $R_3$. These inputs are isolated relay contacts and are "de-bounced" where necessary, in the SDMU, to provide proper system operation. Upon software command the SDMU provides to the MCU, five status bits which come from the five input lines. These are:

1. $D_2$ and/or $D_3$ (2 bits)

2. $R_1$ or $R_2$ or $R_3$ (3 bits)

Light Driver Unit (LDU) -- The SDMU connects to the LDU via 22 data lines. These lines form a 22-bit output data bus, i.e., signal light information, from the SDMU to the load switches. Eleven bits form the main street data while the remaining 11 bits form the cross street data.

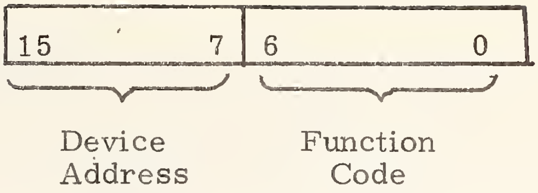
| Line, Main (Cross) | Function |
|---|---|
| 1 (12) | Red primary flow |
| 2 (13) | Yellow primary flow |
| 3 (14) | Green primary flow |
| 4 (15) | Red secondary flow |
| 5 (16) | Yellow secondary flow |
| 6 (17) | Green secondary flow |
| 7 (18) | Don't walk |
| 8 (19) | Walk |
| 9 (20) | Special red |
| 10 (21) | Special yellow |
| 11 (22) | Special green |

Conflict Monitor Unit (CMU) -- Two lines interface the CMU with the SDMU. One line indicates a Conflict and is input from the CMU to the SDMU. The other line is a software generated reset command, which is output from the SDMU to the CMU.

Real-Time Clock Interrupt -- The SDMU contains a 0.1-second period clock with a duty cycle of 50 percent for the purpose of generating an interrupt to the MCU. The clock starts and stops upon command from the MCU.

Flashing "DON'T WALK" -- The SDMU provides a timer which, upon software command, causes the DON'T WALK signal to flash on and off for main or cross street traffic. The ON and OFF times are 0.5 second each (total period equals 1.0 second). The flashing function is implemented downstream of the output data register, i.e., Bit 7 of the output data register is not changed or modified by the SDMU. When Bit 7 = "1", SDMU implements flashing.

Address Format --The SDMU decodes the information appearing on the 16-Address lines of the MCU to determine proper operation of SDMU. Bits 0-6 indicate the Function Code (the operation or function to be performed, i.e., reset conflict monitor, write display register 1, etc.). Bits 7-15 indicate the device address.



The SDMU responds to the five peripheral device addresses. Even though the CMU and BLSMU have their own addresses, the decoding of the address lines is performed in the SDMU since all communication with these devices takes place through the SDMU.

| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | Device |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | LDU |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | BLSMU |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | CMU |
| | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | Real Time Clock |
| | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | Transfer Control Panel |

Function Codes -- Commands to each peripheral device are device dependent, and therefore, are not necessarily common between devices. See the previous paragraph for word structure.

Conflict Monitor --

| Bit | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Command |
|---|---|---|---|---|---|---|---|---|
| | 1 | 0 | 0 | 0 | 0 | 0 | 1 | Reset Conflict Monitor |

Upon receipt of the reset command by the SDMU, the SDMU issues a reset pulse to the CMU which, in turn, removes the "Conflict" output indication.

Bulb and Load Switch Monitor --

| Bit | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Commands | Comments |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 1 | 1 | Read Status Word 1 | Main Street Data |
| | 0 | 0 | 0 | 0 | 1 | 0 | 0 | Read Status Word 2 | Cross Street Data |

The Status Words correspond to bulb and load switch functional information where a Logic One indicates that current is flowing in that bulb circuit. A Read Status Word 1 command causes the main street data to appear on the 16 data lines of the MCU. The data appearing on the 22-bit data bus from the BLSMU is placed on the 16 input data lines of the MCU as shown below.

| Bit-Main | Function |
|---|---|
| 0 | Red primary flow |
| 1 | Yellow primary flow |
| 2 | Green primary flow |
| 3 | Red secondary flow |
| 4 | Yellow secondary flow |
| 5 | Green secondary flow |
| 6 | Don't walk |
| 7 | (Not used) |
| 8 | Walk |
| 9 | Special red |
| 10 | Special yellow |
| 11 | Special green |
| 12 | (Not used) |
| 13 | (Not used) |
| 14 | (Not used) |
| 15 | (Not used) |

Light Driver Unit (LDU) --

| Bit 6 5 4 3 2 1 0 | Commands | Comments |
|---|---|---|
| 0 0 0 0 0 0 1 | Write Command Buffer 1 | Main Street Data |
| 0 0 0 0 0 1 0 | Write Command Buffer 2 | Cross Street Data |
| 0 0 0 0 0 1 1 | Read Output Register 1 | Main Street Data |
| 0 0 0 0 1 0 0 | Read Output Register 2 | Cross Street Data |
| 1 0 0 0 0 0 1 | Reset Watch-dog Timer | |

The SDMU contains two 12-bit buffer registers which receive and store signal light data from the MCU prior to transmission to two 12-bit output registers. Together, the output and buffer registers form a 24-bit data word.

The output registers are capable of being updated on a 24-bit word basis. The update command is decoded by the SDMU from Bit 15 of the Output Command Word which is written into either Write Command Buffer.

Write Command Buffer -- The Write Command Buffer commands cause the SDMU to store the Output Command Word appearing on the MCU output data lines in identified buffer register.

Output Command Word (OCW) -- The Output Command Word contains the bit assignments for the traffic signal display. It is this OCW which eventually controls the states of the LDU.

| Bit | Function |
|---|---|
| 0 | Red primary flow |
| 1 | Yellow primary flow |
| 2 | Green primary flow |
| 3 | Red secondary flow |
| 4 | Yellow secondary flow |
| 5 | Green secondary flow |
| 6 | Don't walk |
| 7 | Flashing Don't Walk |
| 8 | *Walk |
| 9 | Special red |

*Flashing walk for Washington Only.

| Bit | Function |
|---|---|
| 10 | Special yellow |
| 11 | Special green |
| 12 | **Spare |
| 13 | ***Spare |
| 14 | Spare |
| 15 | Update Bit |

** Main Street Flashing Arrow Washington Only.

*** Main Street Green Arrow Washington Only.

Read Output Register -- A Read Output Register command causes the SDMU to place the contents of the selected register on the MCU input data lines. In addition, the data is bit organized identical to the Output Command Word.

Real Time Clock (RTC) --

| Bit 6 5 4 3 2 1 0 | Commands |
|---|---|
| 1 0 0 0 0 1 0 | Start Real Time Clock |
| 1 0 0 0 0 1 1 | Stop Real Time Clock |
| 1 0 0 0 0 0 1 | Terminate RTC Pulse |

These commands are self explanatory.

Transfer Control Panel (TCP) --

| Bit 6 5 4 3 2 1 0 | Commands |
|---|---|
| 1 0 0 0 0 1 1 | Stop BPCU Control |
| 0 0 0 0 0 1 1 | Read Interconnect Data |

The Stop command is OR'ed with the WDT command to cause the SDMU to issue a signal to the TCP, which causes the transfer relays to disconnect the BPCU from the signal lights.

The read command causes the SDMU to place the loop master interconnect data on the MCU input data lines. The bit organization is as follows:

| Bit | | Mnemonic | |
|---|---|---|---|
| 0 | | Sync | |
| 1 | | R1 | |
| 2 | | R2 | |
| 3 | | R3 | |
| 4 | | D2 | |
| 5 | | D3 | |
| 6 | CMU (Conflict) | CMUC "1" = Conflict | |
| 7 | | (Unassigned) | |
| 8 | | (Unassigned) | |
| 9 | | (Unassigned) | |
| 10 | | (Unassigned) | |
| 11 | | (Unassigned) | |
| 12 | Upstream det (Bus/No Bus) | USD | 0" = Bus |
| 13 | Downstream det (Bus/No Bus) | DSD | |
| 14 | | Wait/Control | |
| 15 | | Disconnect | |

All unassigned bits shall be at Logic Zero.

Interface Timing -- See the Applications Manual IMP-16C, Appendix II.

Interface Circuits --

Load Switches -- The SDMU present an open-collector type of drive circuit to the load switches. The open-collector is capable of sinking up to 30 ma dc and withstanding an open circuit voltage of +24 ± 2 vdc. A Logic One corresponds to sinking current from the load switches at a collector-to-emitter voltage of less than +5 vdc.

Loop Master Interconnect -- There are five lines to indicate interconnect data. The input circuits sense contact closure. See Figure 41.

MCU-SDMU --

Address Lines -- All SDMU address lines are TTL compatible and do not present more than one equivalent TTL load per line to the MCU. A Logic Zero corresponds to zero volts.

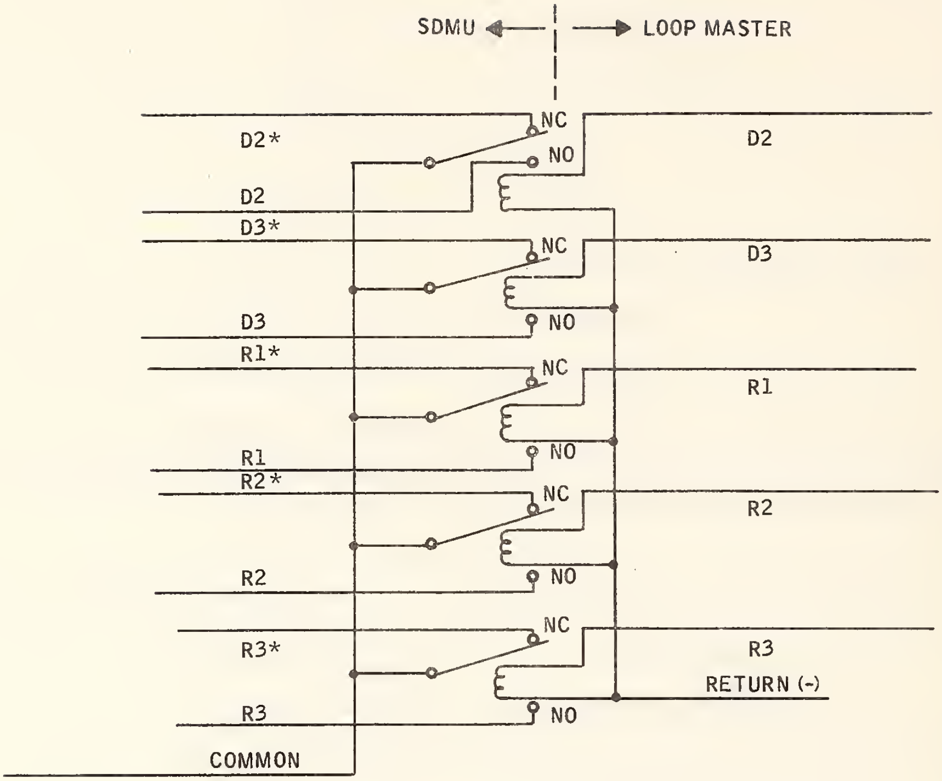


Figure 41. Loop Master Interface

Output Data -- All output data from the MCU to the SDMU is TTL compatible. The SDMU does not present more than one equivalent TTL load per line to the MCU. A Logic Zero corresponds to zero volts.

Input Data -- Input data from the SDMU is placed upon the data lines with TTL-compatible, tri-state logic. A Logic Zero corresponds to zero volts.

Conflict Monitor Unit --

Reset Conflict Monitor Line -- The output voltage to the CMU is TTL compatible. A Logic One corresponds to 1.0 v.

Conflict Indication -- The SDMU circuit which receives the conflict indication from the CMU is TTL compatible. The SDMU does not present more than one equivalent TTL load to the CMU. A Logic One is indicated by an open circuit.

Hardware Description --

Table 7 identifies the major components in the SDMU, sheet-by-sheet of the schematic.

Table 7. SDMU Major Components

| Sheet | IC Location | Description |
|---|---|---|
| 1 | 1R-P | Peripheral function decoder |
| | 1P-N | Peripheral address decoder |
| 2 | 1F, 1G, 2F, 3F, 4F, 5F | Data buffers |
| | 5E, 4E, 3E | Status display register |
| | 4D, 5D | Status display latch |
| | 4J, 4H | Main Street data buffer register |
| | 5J, 5H | Main Street data output register |
| | 4L, 4K | Cross Street data buffer register |
| | 5L, 5K | Cross Street data output register |
| 3 | 1L, 2L, 1H, 2H, 1J, 2J, 1K, 2K | Peripheral input data drivers |
| | 2M, 3M, 3L, 3K, 3J, 3H | Input data multiplexers |
| 4 | 5A, 4A, 3B, 3A, 2A, 2B, 1B, 1C | 0.1 sec interrupt generator |
| | 5R, 5P | Flash function generator |
| | 6F, 6D, 6J, 6H, 6L, 6K, 6E | Solid state relay drivers |
| 5 | 6C, 4D | Loop master data input buffers |
| | 5G, 4D | Watchdog timer |
| 6 | 1E, 5N | Main Street data buffer and output registers |

Signal Names -- Table 8 presents the most significant names for understanding signal flow.

Direct Memory Access (Cycle Steal)

The cycle steal device is a Honeywell-developed addition to the National Semiconductor IMP-16 microprocessor. In essence, the Cycle Steal is a simplified direct memory access (DMA) device.

Well into the program, after more information was gathered about the characteristics of the preprocessor, it was apparent that the relatively high

Table 8. Significant Names

| Signal | Function | Sheet |
|---|---|---|
| ABOO-15 | Peripheral Address Lines | 1 |
| RDP | Read Peripheral Data | 1 |
| INIT* | Initialization | 1 |
| WRP | Write Peripheral Data | 1 |
| RMSD | Read Main Street Data | 1 |
| RCSD | Read Cross Street Data | 1 |
| RSW1 | Read Status Word One (BLSMU) | 1 |
| RSW2 | Read Status Word Two (BLSMU) | 1 |
| RWDT | Reset Watch-dog Timer | 1, 4 |
| WSDD | Write Status Display Data | 1, 2 |
| SMCUC | μP Command Disconnect | 1, 5 |
| STRTC | Start Real Time Clock | 1, 4 |
| STPRTC | Stop Real Time Clock | 1, 4 |
| CMRS | Conflict Monitor Reset | 1 |
| BDOO-15 | Buffered Data (Peripheral) | 2 |
| SDOO-15 | Status Display Data | 2, 4 |
| MSDOO-11 | Main Street Data | 2, 3, 4 |
| CSDOO-11 | Cross Sheet Data | 2, 3, 4 |
| SWOO-15 | Peripheral Data | 3 |
| INTRA | Interrupt | 4 |
| R1S | Offset 1 | 4 |
| R2S | Offset 2 | 4 |
| R3S | Offset 3 | 4 |
| D2S | Dial 2 | 4 |
| D3S | Dial 3 | 4 |
| DBPCU | Disconnect BPCU | 4 |
| MSD12-13 | Main Street Data | 6 |

data rates which could occur, worst case, between the preprocessor and the IMP-16 would be disastrous to the execution of the signal timing plan if the I/O bus of the IMP-16 were used for the data transfer. An exact data rate is impossible to theorize because it depends upon the situation being sensed by the loop electronics. This includes vehicle type as well as possible noise sources. However, one can obtain a qualitative idea from the following calculation.

A = Number of detectors (max) = 16

B = Number of data per signature feature = 2

C = Additional data per signature = 4

D = Number of features per signature - 1 minimum, ? maximum, 2 typical

$$\{ (BxD) + C \} A = \text{Data Quantity (DQ)}$$

$$DQ = \{ (2x?) + 4 \} 16$$

$$DO \min = 96 \text{ words}$$

Another fact is that the incoming data is completely asynchronous with the CPU operations.

If we also assume that vehicle speeds are 55 mph and an average vehicle length is 17 feet, then 16 signatures can arrive in:

$$\frac{x}{55} = \frac{88}{60}$$

$$x \text{ ft/sec} = \frac{55}{60}(88) = 80.66 \text{ ft/sec}$$

$$T = \frac{17 \text{ feet}}{x \text{ ft/sec}} = 0.210 \text{ sec}$$

Therefore, the minimum data rate is:

$$DR = \frac{DO}{T} = \frac{96}{0.21}$$

$$DR_{min} \approx 457 \text{ words/sec}$$

and $$DR_{typ} = \frac{128}{0.21}$$

$$DR_{typ} \approx 610 \text{ words/sec}$$

Table 9 contains the alternatives considered and the rationale behind the decision for choosing the Cycle Steal approach.

Table 9. Data Rate Tradeoff

| Alternatives | Resolution |
|---|---|
| Transfer data from the preprocessor to the μP via the I/O bus by (1) generating an interrupt when data is to be transferred, or (2) poll the preprocessor at prescribed intervals. | Rejected. Since the CPU must service the signal head timing plan at the same time and must do so at 0.1 sec. intervals, both methods require too much software overhead at these data rates |
| Temporarily store data in a peripheral until the μP is ready to read and process the data. Use the I/O bus. | Rejected. This method requires a duplication of what already exists in the μP, namely, dedicated memory. It also requires relatively complex control logic between the real-time preprocessor and the μP. |
| Use a DMA kind of device to store data, as it comes, into the microprocessor memory. | Accepted. This method frees the I/O bus for signal head timing plan execution. It also requires minimum software overhead. And, it does not duplicate existing system components. |

Figure 42 is a block diagram of the Cycle Steal device. The operation is, simply, that the preprocessor asynchronously places address and data information into the appropriate registers of the Cycle Steal and then commands the timing and control logic to initiate the memory storage action. No further action is required by the preprocessor. The timing and control logic monitors the IMP-16 to see when the memory is not in use and then stores the data at the specified address. The CPU is not affected by this operation and thus no CPU or software overhead is incurred.

The timing of the Cycle Steal is identical with, and tied to, the IMP-16. Detailed timing can be found in the IMP-16 description.

The Cycle Steal is designed to operate with static and dynamic RAM. The refresh logic is disabled in the BPCU which uses static RAM. The device was developed in a system which used dynamic RAM, hence, the reason it exists at all.

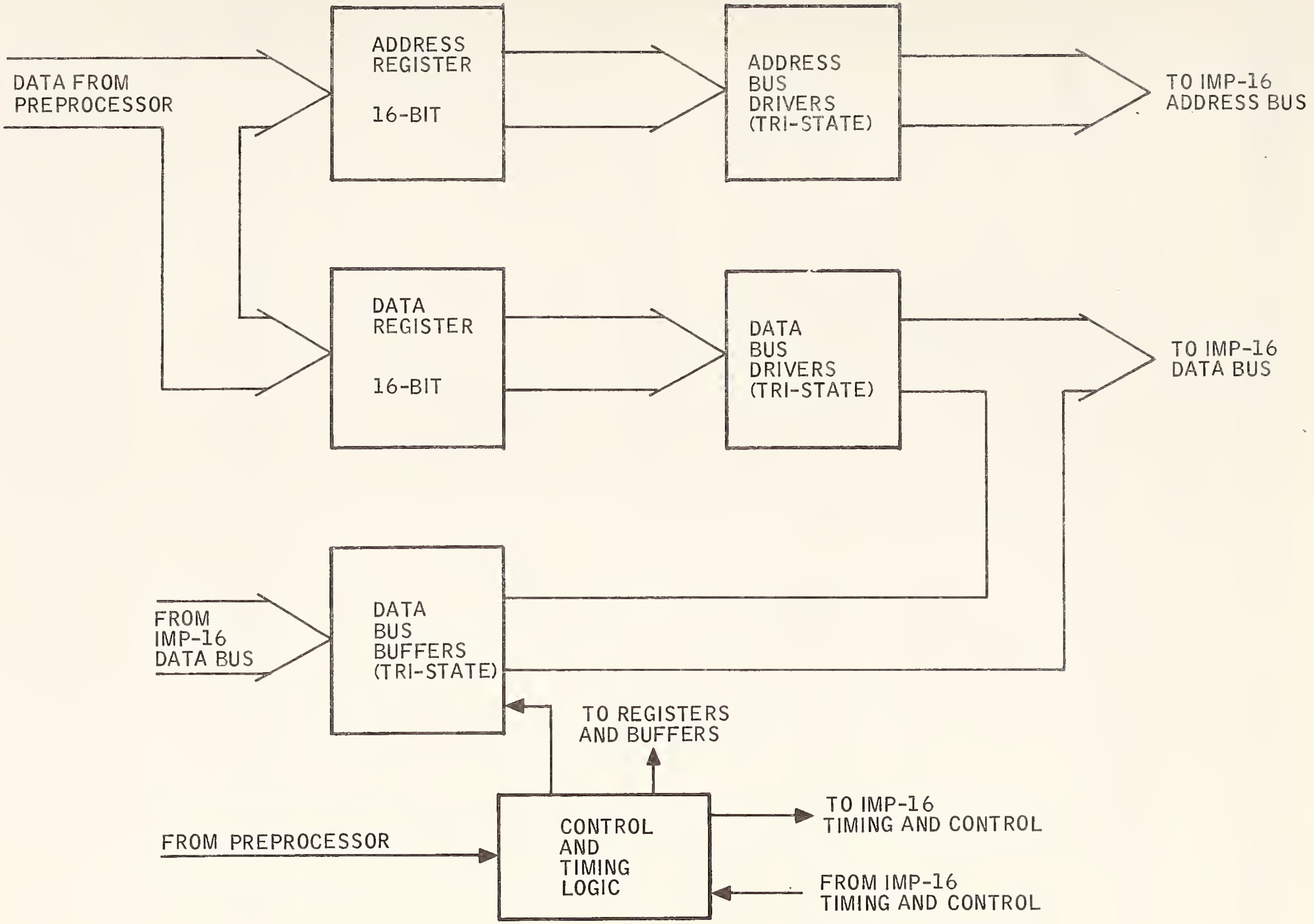


Figure 42. Cycle Steal Block Diagram

Hardware Description -- Table 10 identifies the major components in the Cycle Steal, sheet-by-sheet of the schematic.

Table 10. Cycle Steal Major Components

| Sheet | IC Location | Description |
|---|---|---|
| 1 | 1P, 2P, 2J | IMP-16 memory monitor and output buffer control |
| | 2P, 3J | Memory write strobe control |
| | 2N, 2K | Clock stretching control |
| | 2R, 2M, 1P | 8-phase timing generator |
| 2 | 3R, 3P | Preprocessor, steal request |
| | 3P | Refresh request memory (for dynamic RAM) |
| | 1N, 1L, 1M | Tri-state address bus drivers |
| 3 | 1F, 1H, 1K | IMP-16 data bus drivers (tri-state) |
| | 1E, 1G, 1J | Cycle Steal data bus drivers (tri-state) |
| 4 | 2E, 2F | Cycle Steal data register |
| | 2H, 3H | Cycle Steal address register |

Signal Names -- The following names (Table 11) are the most significant ones for understanding signal flow.

Table 11. Signal Flow Significant Names

| Name | Description | Sheet |
|---|---|---|
| PPSTEAL* | Preprocessor steal request | 1 |
| RDP | Read peripheral data | 1 |
| WRP | Write peripheral data | 1 |
| RDM | Read memory data | 1 |
| WRM | Write memory data | 1 |
| EXHOLD | Clock hold request (external) | 1 |
| CLK* | Master CPU clock (5 MHz) | 1 |
| CIS | Cycle initiate - steal | 1 |
| ODIS | Output disable - (CPU address drivers) | 1 |
| WRMS | Write memory data - steal | 1 |
| RFREQ | Refresh request | 2 |
| PPSTL* | Preprocessor steal request enable | 2 |
| PPSTLP* | Preprocessor steal request strobe | 2 |
| ADP00-15 | Address register data | 2,4 |
| ADX00-15 | IMP-16 address bus | 2 |
| BDO00-15 | IMP-16 data bus | 3 |
| BDO00S-15s | Multiplexed IMP-16 data bus | 3 |
| PDO00-15 | Preprocessor data register output | 4,3 |
| BPPD00-15 | Preprocessor data bus | 4 |
| CS0 | Chip select zero | 4 |
| CS1 | Chip select one | 4 |
| CS2 | Chip select two | 4 |

## Programmable Panel

The programming panel is a device which allows the traffic engineer to make certain limited changes in the timing plan for the traffic signals. It is designed specifically for use with the Bus Priority System.

Applicable Sections --

- Application Manual IMP-16 (National Semiconductor)
- Status Display and Monitor Unit (SDMU)

General Description -- The programming panel subsystem is divided into two components; the operator panel containing the switches and indicator lights, and the interface electronics which convert the switch data to the proper format for use by the MCU. See Figure 43.

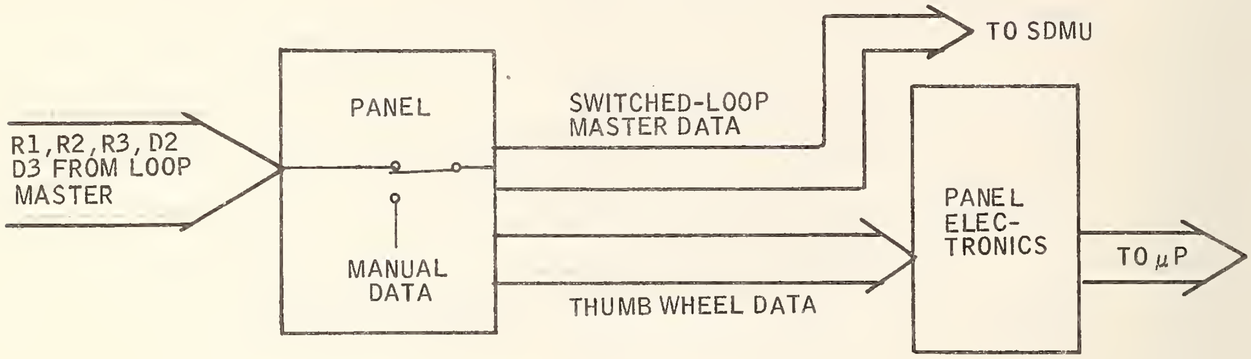


Figure 43. Programming Panel Block Diagram

The programming panel is treated as a peripheral device(s) by the microprocessor.

The panel also intercepts the loop master data and is capable of manually overriding the data (Figure 44). Thus, any one of the three timing plans on the front may be selected manually or automatically. The remaining switches are self-explanatory.

Detailed Functional Description --

Interface Electronics -- The Programming Panel decodes the information appearing on the 16 address lines of the MCU to determine the proper operation for this device. Bits 0-6 indicate the Function Code (the operation or function to be performed).

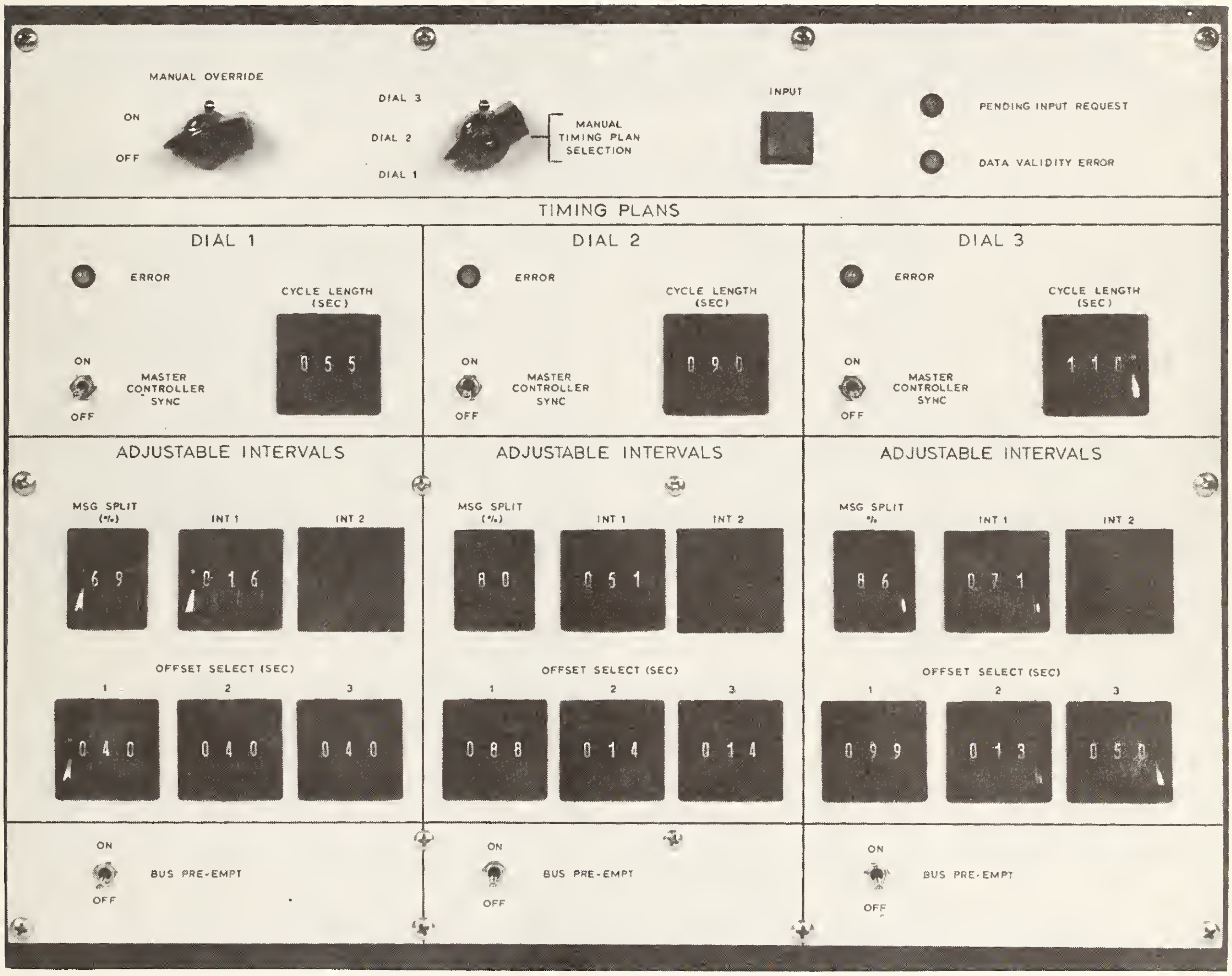


Figure 44. Programming Panel

Bits 7-15 indicate the peripheral device address.

| 15 7 | 6 0 |
|---|---|
| Device Address | Function Code |

Peripheral Devices -- The Programming Panel is identified by four peripheral addresses, one for each timing plan and one for the various On-Off status switches on the front of the panel.

|  | Peripheral Address |  |  |  |  |  |  |  |  | Device |
|---|---|---|---|---|---|---|---|---|---|---|
| Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |  |
|  | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | Timing Plan 1 |
|  | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | Timing Plan 2 |
|  | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | Timing Plan 3 |
|  | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | Status Word |

The Timing Plan Selection switch is not addressed by these electronics.

Status Word -- The status word contains the toggle switch information for the entire panel. All unused bits are set to zero on the interface. The ON position indicates a logic ONE.

| Bit | Switch |
|---|---|
| 0 | Bus Pre-empt, Plan 1 |
| 1 | Master Controller Synch, Plan 1 |
| 2 | --- |
| 3 | --- |
| 4 | Bus Pre-empt, Plan 2 |
| 5 | Master Controller Synch, Plan 2 |
| 6 | --- |
| 7 | --- |
| 8 | Bus Pre-empt, Plan 3 |
| 9 | Master Controller Synch, Plan 3 |
| 10 | Input Request Pushbutton |
| 11 | Manual Override |
| 12 | --- |
| 13 | --- |
| 14 | --- |
| 15 | --- |

Function Code --

Timing Plan -- The following function codes apply to the timing plan peripheral addresses.

| | Function Code | | | | | | | Function |
|---|---|---|---|---|---|---|---|---|
| Bits | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | 0 | 0 | 1 | 0 | 0 | 0 | 1 | Read Cycle Length |
| | 0 | 0 | 1 | 0 | 0 | 1 | 0 | Read Offset Select 1 |
| | 0 | 0 | 1 | 0 | 0 | 1 | 1 | Read Offset Select 2 |
| | 0 | 0 | 1 | 0 | 1 | 0 | 0 | Read Offset Select 3 |
| | 0 | 0 | 1 | 0 | 1 | 0 | 1 | Read MSG Split |
| | 0 | 0 | 1 | 0 | 0 | 0 | 0 | Read Interval 1 |
| | 0 | 0 | 1 | 0 | 1 | 1 | 1 | Read Interval 2 |
| | 1 | 0 | 0 | 0 | 1 | 0 | 0 | Turn On Status Light |
| | 1 | 0 | 0 | 0 | 1 | 0 | 1 | Turn Off Status Light |

Status Word -- The status word data is presented to the peripheral data lines when the status word address and the following function code are presented to the panel electronics.

| | Function Code | | | | | | | Function |
|---|---|---|---|---|---|---|---|---|
| Bits | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 1 | 1 | Read Discrete Status Word |

Thumbwheel Switch Data -- The Thumbwheel data is organized as a BCD word with the following format. This data comes from the switches so no reformatting is necessary. The format below, therefore, constitutes one complete thumbwheel setting, i. e. , cycle length, interval, etc.

| 15 | 12 | 11 | 8 | 7 | 4 | 3 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | BCD | | BCD | | BCD | |
| | | Hundreds Digit<br>(where applicable) | | Tens Digit | | Units Digit | |

Input Request indicator is lighted by the interface electronics when the input request pushbutton is pressed. The indicator remains lit until the button is pressed again or reset by a "turn off status light" function code to the status word peripheral address.

Panel -- For the most part, the switching functions performed by the discrete switches are self explanatory. The following descriptions provide additional explanation for certain switches.

Thumbwheel Switches -- Each digit of a particular switch function has a BCD output corresponding to the decimal indicated value.

Manual Timing Plan Selection -- These switches directly affect the loop Master data; D2, D3, R1, R2, and R3. When the manual override switch is OFF, these signals are passed to the SDMU without modification.

When the override switch is ON, the following conditions are generated:

1) R1, R2, and R3 are set to zero.

2) A signal is sent to the panel electronics, for use in the status word, indicating that the override switch is ON.

3) D2 and D3 are controlled by the timing plan selector switch.

Timing Plan Selector Switch -- This switch, when enabled by the override switch, generates the following dial select code.

| SW. Position | D2 | D3 |
|---|---|---|
| P1 | 0 | 0 |
| P2 | 1 | 0 |
| P3 | 0 | 1 |

## Panel Electronics - μP Interface

Timing and logic levels conform to the IMP-16 I/O bus.

Hardware Description -- Table 12 identifies the major components in the panel electronics of the schematic sheet-by-sheet.

Table 12. Panel Electronics Major Components

| Sheet | IC Location | Description |
|---|---|---|
| 1 | 1G-f | Peripheral address decode |
| 2 | 1H1J, 1J-K | Function code decoders |
| 3 | 1M<br>2A, 2B | Peripheral data drive<br>Plan timing flip flops |
| 4 | 6P, 6R | Switch debounce flip flops |
| 5 | 6P, 3J, 2C<br>2C, 6C, 1D | Input Request Pushbutton Logic<br>Input Request Indicator Logic |
| 6 | 3R, 4R, 5R | Data Multiplexers |
| 7 | 3P, 4P, 5P | Data Multiplexers |
| 8 | 3N, 4N, 5N | Data Multiplexers |
| 9 | 3L, 4L, 5L | Data Multiplexers |
| 10 | 3K, 4K, 5K | Data Multiplexers |
| 11 | 3H, 4H, 5H | Data Multiplexers |
| 12 | 3G, 4G, 5G | Data Multiplexers |
| 13 | 3E, 4E, 5E | Data Multiplexers |
| 14 | 3D, 4D, 5D | Data Multiplexers |
| 15 | 3B, 4B, 5B | Data Multiplexers |
| 16 | 3A, 4A, 5A | Data Multiplexers |

Signal Names -- Table 13 lists the most significant names for understanding the signal flow.

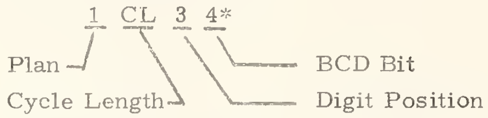
Power Supplies -- The requirements for power supplies for the BPCLU were based upon calculations and upon current-drain measurements where possible. The IMP-16 is a combination of MOS and TTL technology and, therefore, requires +5vdc and -12vdc. In addition, the RAM memory on the CPU card requires -9vdc. We elected to use the CPU RAM because the CPU is manufactured and sold that way.

The preprocessor uses ±15vdc in addition to +5vdc. The analog multiplex and A/D converter require ±15vdc. The TTL requires the +5vdc.

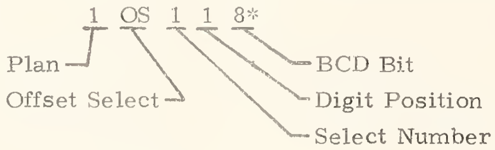
Table 14 contains the power requirements for the logic cards. The +5vdc supply is of particular interest. Table 14 shows that the BPCU requires an estimated 16 amps typical. If we allow a 50 percent safety margin, our maximum current rises to 24 amps at +5vdc. The supplies which were readily available to the program were rated at 25 amps and 35 amps. For obvious reasons, we chose the supply rated at 35 amps.

## Table 13. Significant Names

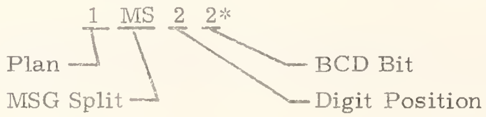
| Signal | Function | Sheet |
|---|---|---|
| RDP | Read Peripheral | 1 |
| T4* | Clock | 1 |
| INIT | Initialization | 1 |
| WRP | Write Peripheral Data | 1 |
| AB07-15 | Peripheral Address Lines | 1 |
| TIME 3* | Timing Plan 3 Address Decode | 1 |
| TIME 2* | Timing Plan 2 Address Decode | 1 |
| TIME 1* | Timing Plan 1 Address Decode | 1 |
| STATW* | Status Word Address Decode | 1 |
| AB00-06 | Peripheral Address Lines | 2 |
| RDI2* | Read Interval 2 Data Function | 2 |
| RDI1* | Read Interval 1 Data Function | 2 |
| RDMS* | Read Main Street Split Data | 2 |
| RDOS3* | Read Offset 3 Function Decode | 2 |
| RDOS2* | Read Offset 2 Function Decode | 2 |
| RDOS1* | Read Offset 1 Function Decode | 2 |
| RDCL* | Read Cycle Length Function Code | 2 |
| RDDSW* | Read Discrete Status Word | 2 |
| TOFDEL* | Data Validity Error Light Control | 2 |
| TONDEL* | Data Validity Error Light Control | 2 |
| TONSL* | Turn-on Status Light | 2 |
| TONSL* | Turn-off Status Light | 2 |
| SW12-15 | Peripheral Data Lines | 3 |
| TIM1L* | Plan 1 Error Light Control | 3 |
| TIM2L* | Plan 2 Error Light Control | 3 |
| TIM3L* | Plan 3 Error Light Control | 3 |
| BPE1*-3* | Pre-empt Switch Status | 4 |
| MANO | Manual Override Switch Status | 4 |
| MCS1*-3* | Master Controller Switch Status | 4 |
| PIRL* | Input Request Light Control | 5 |
| DVEL* | Data Validity Error Light Control | 5 |
| SW00-11 | Peripheral Data Lines | 6-16 |



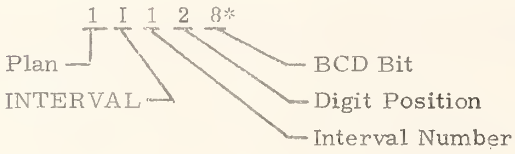
6, 10, 13



7, 10, 11
14, 15



8, 12, 15



9, 12, 13,
16

Table 14. BPCU Power Requirements

| Card | Amps @ +5v | Amps @ -12v | Miscellaneous |
|---|---|---|---|
| CPU IMP-16C/300 | 2.25(specified) | 0.5A(specified) | 0.6A at -9V |
| 4K RAM IMP-16P/004 | 4.0 max<br>2.0 typ | | |
| 4K PROM Special -22 | 1.25A max half loaded<br>1.9A max full load | 1.3A max full load<br>0.9A max half loaded | |
| Debug - 16C, Daynl | 2A typ(estimate) | 100ma(estimated) | |
| Cycle Steal Processor | 1A typ(estimate)<br>3A typ(measured) | | 46ma +15v<br>26ma -15v |
| SDMU | 4.2A max<br>2.5A typ | | |
| Programming Panel | 2A typ(estimate) | | |
| Total: | 16A type(estimate) | | |

All the power supplies for the BPCU were purchased from outside vendors. Table 15 contains the pertinent information. The -9vdc was provided by a simple regulator circuit which was developed by Honeywell for the bus detector program (Figure 45). The voltage stability of this regulator as required by the IMP-16 CPU is not critical. A ±5 percent stability is sufficient. Figure 46 shows the power interconnection for the BPCU.

Table 16 shows the location of each electronics card in the BPCU chassis. Table 17 is a list of all the major subassemblies which make up the BPCU.

## Table 15. Power Supply Specifications

| +5 VDC Supply, Power-One Inc., Model G5-35/OVP | | | |
|---|---|---|---|
| AC Input | 115/230 VAC ± 10%, 47-440 Hz (Derate current 10% for Hz operation | Remote sensing | Provided, open lead protection built in |
| DC Output | 5V ± 5% output adjustment range minimum | Stability | ±0.05% for 24 hours after warm up |
| Line Regulation | ±0.01% for a 10% input change | Temperature rating | 0 to 50°C full rated, derated linearly to 40% at 70°C |
| Load Regulation | ±0.01% ± 1 mv for a 50% load change | Temperature coefficient | ±0.01%/°C maximum, 0.002%/°C typical |
| Output Ripple | 1.5 mv PK-PK, 0.2 mv RMS maximum | Efficiency | 52-54% |
| Transient Response | 30 μseconds for 50% load change | Vibration | Per Mil-Std-801B. Method 514. Procedure 1, curve AB (to 50 Hz) |
| Short Circuit and Overload Protection | Automatic current limit/ foldback | Shock | Per Mil-Std-801B, Method 516, Procedure V |
| OVP | Built-in at 6.2 ± 0.2 VDC | Master/slave | Any unit may be master or slave. One master will drive up to 5 slaves |
| Current Output | 35 amps | | |

| ±15 VDC Supply, Power-One Inc., Model AA15-0.8 | | | |
|---|---|---|---|
| AC Input | 105-125 VAC, 47-440 Hz (Derate current 10% for 50 Hz operation) | Remote sensing | Provided, open lead protection built-in |
| DC Output | ±15V ± 5% output adjustment range minimum | Tracking accuracy | ±0.1% when balance is present to ±0.01% |
| Line Regulation | ±0.01% for a 10% input change | Stability | ±0.05% for 24 hours after warm up |
| Load Regulation | ±0.02% for a 50% load change | Temperature rating | 0 to 50°C full rated, derated linearly to 50% at 70°C |
| Output Ripple | 1.5 mk PK-PK, 0.4 mv RMS maximum | Temperature Coeffieicnt | ±0.01%/°C maximum, 0.002% typical |
| Transient Response | 30 μseconds for 50% load change | Efficiency | 55% |
| Short Circuit and Overload Protection | Automatic current limit/foldback | Vibration | Per Mil-Std-810B, Method 514, Procedure 1, Curve AB (to 50 Hz) |
| Reverse Voltage Protection | Provided on output and pass element | Shock | Per Mil-Std-810B, Method 516, Procedure V |
| Current Output | 800 ma each output | | |

| -12 VDC Supply, Power-One Inc., Model C15-3 | | | |
|---|---|---|---|
| AC Input | 105-125 VAC, 47-440 Hz (Derate current 10% for 50 Hz operation) | Remote sensing | Provided, open lead protection built-in |
| DC Output | See table, ±5% output adjustment range minimum | Stability | ±0.05% for 24 hours after warm up |
| Line Regulation | ± 0.01% for a 10% input change | Temperature rating | 0 to 50°C full rated, derated linearly to 40% at 70°C |
| Load Regulation | ± 0.02% for 50% load change | Temperature coefficient | ±0.01%/°C maximum, 0.002% typical |
| Output Ripple | 1.5 mv PK-PK, 0.4 mv RMS maximum | Efficiency | 55% |
| Transient Response | 30 μseconds for 50% load change | Vibration | Per Mil-Std-810B, Method 514, Procedure 1, Curve AB (to 50 Hz) |
| Short Circuit and Overload Protection | Automatic current limit/foldback | Shock | Per Mil-Std-810B, Method 516, Procedure V |
| Reverse Voltage Protection | Provided on output and pass element | | |
| Current Output | 3.4 amps | | |

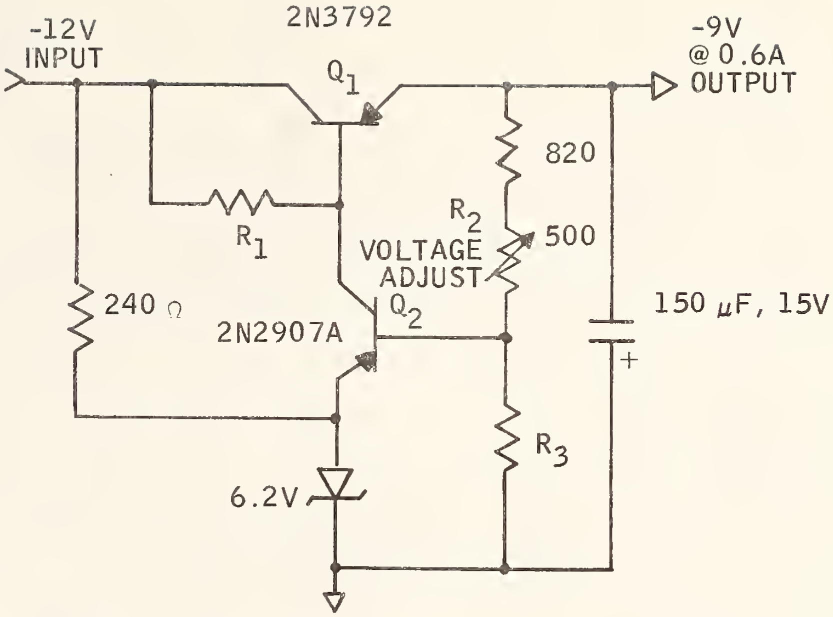


Figure 45. -9.0-Volt Regulator

Table 16. Card Location Assignments (Top View)

| Connector | Electronics Cord | |
|---|---|---|
| J10 | Computer Control Panel | Rear |
| J9 | 4K RAM | |
| J8 | 4K PROM | |
| J7 | CPU | Component Side |
| J6 | --- | |
| J5 | Cycle Steal | |
| J4 | SDMU | |
| J3 | Preprocessor No. 2 | |
| J2 | Preprocessor No. 1 | |
| J1 | Programming Panel | |

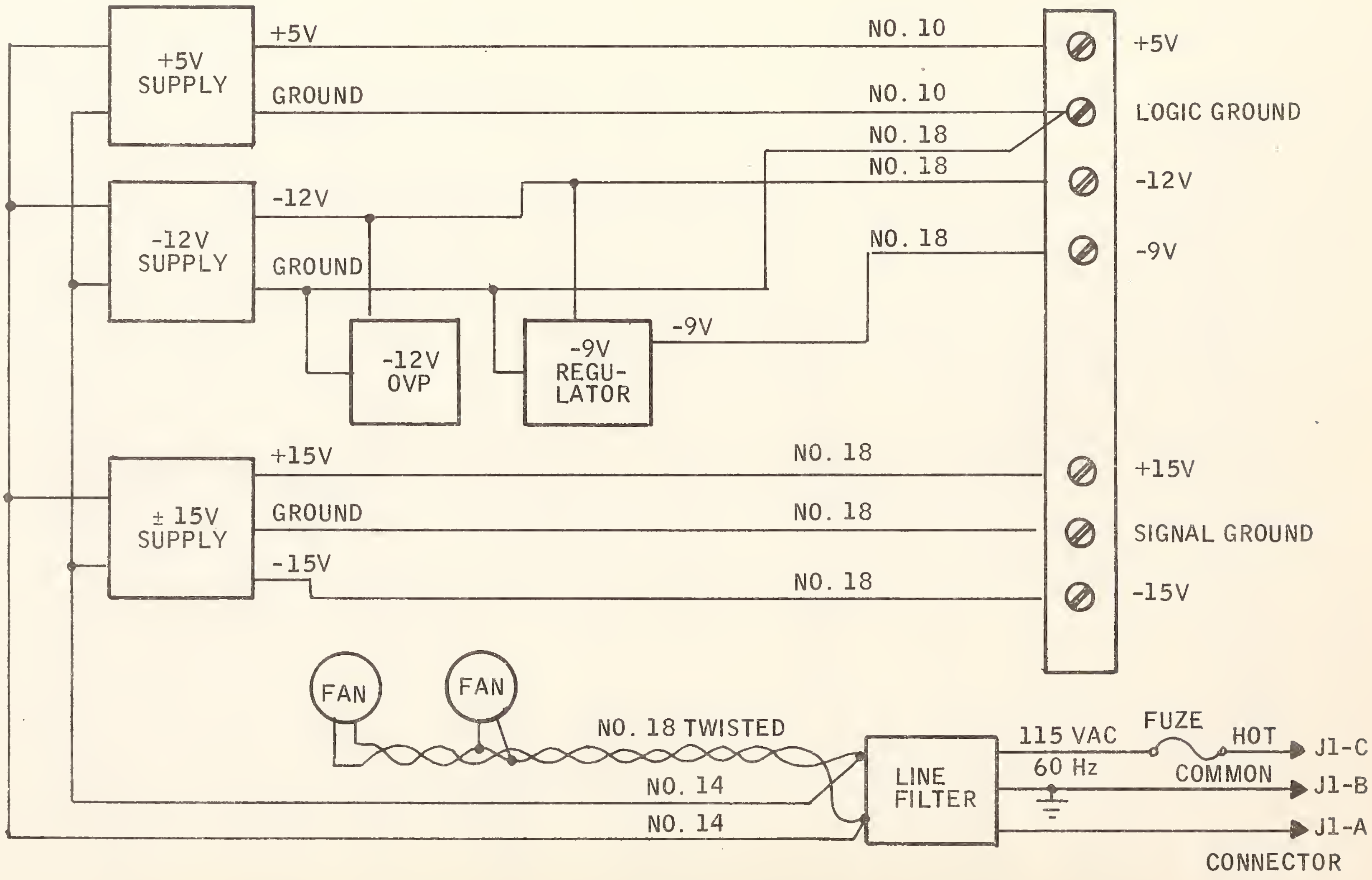


Figure 46. BPCU Power Schematic

Table 17. BPCU Subassemblies

| Quantity | Description |
|---|---|
| 1 | Chassis, Honeywell |
| 1 | Card Cage, 6-Slot; National Semiconductor (NS), IMP-OOH/880 |
| 2 | Card Cage, 3-Slot; NS, IMP-OOH/881 |
| 2 | Fan; Boxer, Model BS2107F-0-1 |
| 1 | Microprocessor; NS, IMP-16C/300 |
| 1 | PROM Memory Card; NS, IMP-Special 22 |
| 1 | RAM Memory Card; NS, IMP-16P/004A |
| 1 | Cycle Steal Card; Honeywell |
| 1 | Status Display and Monitor Unit (SDMU); Honeywell |
| 1 | Programmable Panel; Honeywell |
| 1 | Programmable Panel Electronics Card; Honeywell |
| 1 | Preprocessor Card Set; Honeywell |
| 1 | Debug Aid; Daynl, DEBUG-16/C |
| 1 | Power Supply, +5V, 35A; Power-One, Model G5-35/OVP |
| 1 | Power Supply, -12V, 3.4A; Power-One, Model C15-3 |
| 1 | Power Supply, ±15V, 0.8A; Power-One, Model AA15-0.8 |
| 1 | Regulator, -9V, 0.6A; Honeywell |
| 1 | RFI Filter; Cornell-Dubiliar, NFR113-9 |
| 2 | Connector; MS 3102A-36-10 |
| 2 | Connector; MS 3102A-28-21 |

## System Conflict Monitor and Ancillary Equipment

### Conflict Monitor

Functional Description -- The Conflict Monitor is a device that provides green and walk conflict monitoring, minimum green and minimum yellow interval monitoring, and maximum cycle length monitoring. If the controller fails to satisfy these criteria, the Conflict Monitor will present to the controller a signal to cause the transfer of intersection control to flashing operation. Once flashing operation has been initiated, a reset, either from a remote source or from a front panel push button, is required to return the controller from flashing operation. A block diagram of the unit is shown in Figure 47.

The Conflict Monitor samples the 115-volt vac output of the solid-state load relays. This is the same line used to drive the lamps in the signal heads. From this signal, the Conflict Monitor can monitor any failure in the system that would cause improper or dangerous displays at the signal head. The 115-volt vac sense circuit shown in the block diagram presents to the internal circuitry a logic level signal displaying the presence of 115 volts on the lamp drive circuits.

The green conflict monitor portion of the unit will provide monitoring of nine signal circuits. The circuits are configured for four phases, two with concurrent pedestrian movements, and three overlap phases. Only one of the four sequential phases is allowed to be on at any one time. Each overlap phase is programmable by diode-matrix, to be conflicting with any or all of the other phase green signals.

An internal clock is used for timing minimum intervals and maximum cycle lengths. The clock is based on 60-cycle line frequency. Two sets of minimum green and minimum yellow timing elements are used. One set is used for the four sequential phases. The second set is used for those phases which are "on" concurrently with one or more of the four sequential phases. If any one of the green or yellow signal is removed before the timing element completes its timing, then a latched conflict signal is generated. In the event that a green or yellow is skipped entirely, the number of greens or yellows is compared to an internally preset value at the end of each cycle. If the actual number of greens or yellows during each cycle does not agree with the preset number, a latched conflict indication is generated.

Each background cycle beginning with phase three green, the maximum cycle length counter, begins counting down from its preset, thumbwheel switch selected value. If the counter is allowed to time out before a phase three green again presets the counter, a latched conflict indication is generated.

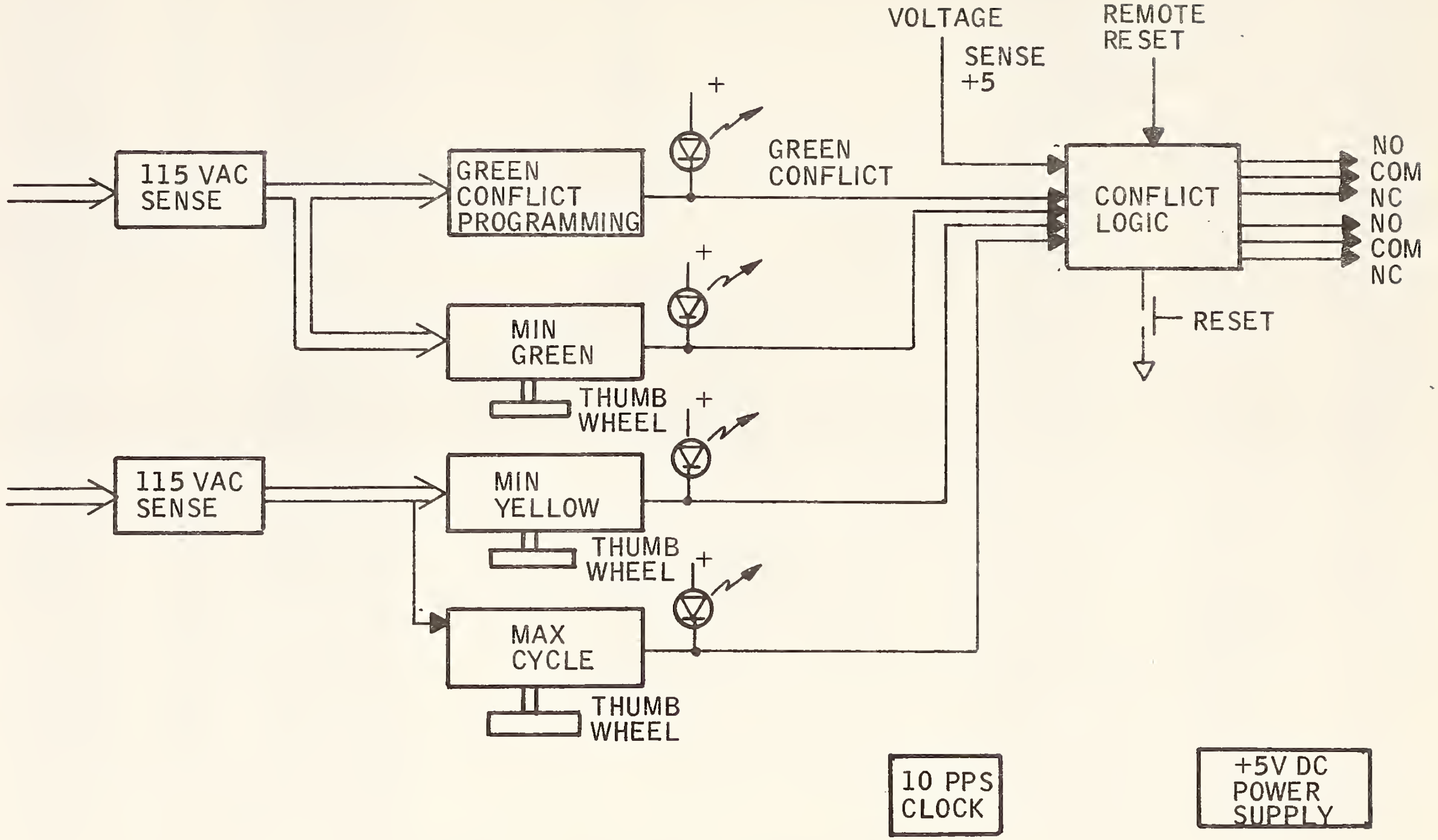


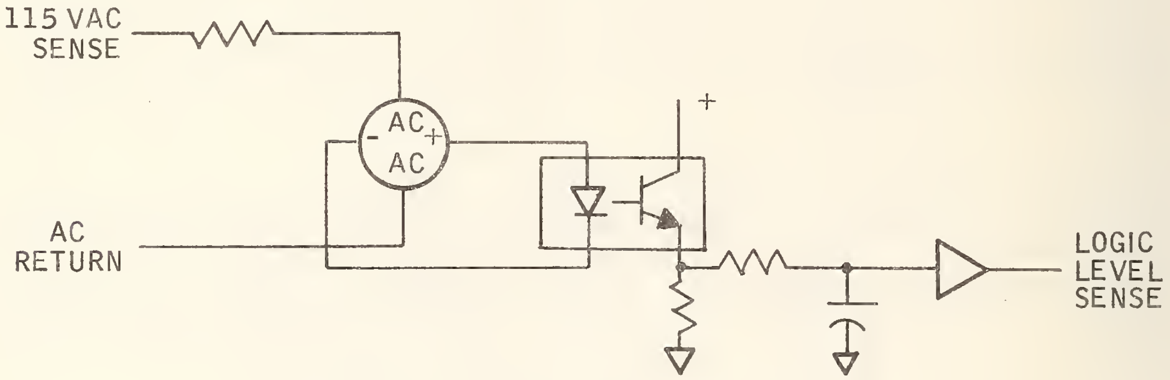
Figure 47. Operational-Conflict Monitor

A 5 vdc sense input is also provided to monitor controller logic power. If 5 vdc is not preset, the unit will place the intersection in flash. Should 5 vdc be reapplied, the conflict monitor will be reset and flashing operation will terminate.

The outputs from the Conflict Monitor are the contacts of a double-pole relay. The relay is held energized unless a conflict occurs or power is removed from the Conflict Monitor.

Circuit Description --

115 VAC Sense -- The sense circuits are designed to detect approximately 20 vrms on either the positive or negative half wave. All inputs are optically isolated and are sufficiently filtered to prevent transients on input circuits from tripping the monitor.



Conflict Logic -- The outputs of the 115 VAC circuits are combined in a logic array to compare each input with all other inputs. Any two lamps which should not be on at the same time will cause a conflict signal to be generated. If the signals remain in conflict for longer then 250 msec, then the conflict signal is latched into memory and the conflict state is presented to the controller.

Timing Circuits -- The timing circuits are implemented as shown in Figure 48.

If any one of the phase greens turns on, then the two divide by 10 and counting stages (A & B) begin timing down from their preset value. A 10-pulse-per-second clock was used rather than a 1-pulse-per-second clock to avoid

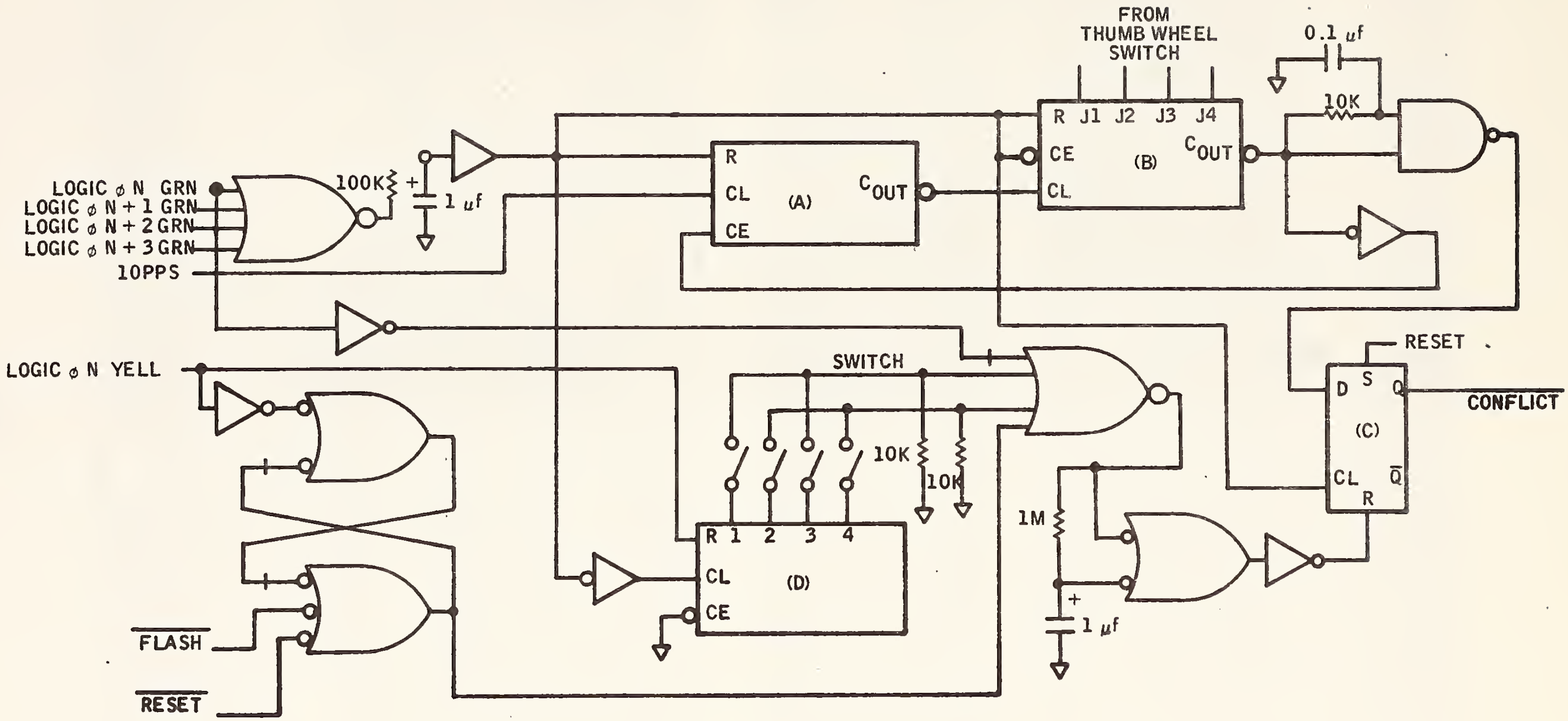


Figure 48. Timing Circuit Implementation

synchronization problems at this controller-conflict monitor interface. When the phase green terminates, the flip-flop (C) is clocked. Unless the counters have timed out and are presenting a logic "1" at the D input of the flip-flop, a conflict signal is generated.

The third counter (D) is used to count the number of times a phase green occurs. Each time an ON green occurs the counter is sampled. Unless the proper number of greens, as selected by the switches, has occurred, a signal is presented to the reset input of the flip-flop (C) to cause a conflict. The counter is reset at the application of ON yellow. The counter thus counts the number of greens that occur between each ON yellow. Each time that the unit is reset or that the unit recovers from a flashed operation, the number of greens is not sampled for one full cycle. This allows the monitor to be sure that a full cycle of counts has been accumulated. There are four of these timer circuits, one each for two sets of green and yellow circuits.

Cabinet -- The cabinet used in the system is a type P-1. This cabinet is completely weather proof and of rugged construction, welded from No. 14-gauge sheet steel, primed and enameled, providing for the greatest possible corrosion resistance. The exterior design includes a slanted roof and drip-proof shelf. The P-1 door is full width of the cabinet and includes a lock securing a three-point latch. The three-point latch consists of two rods holding the top and bottom in conjunction with a standard rumbler lock to provide protection against tampering and weather. All three holding points operate from one large easy turning handle.

No bolts are required to close or lock the P-1 cabinet door. A single-position door stop holds the door open to an approximate 90-150-180 deg to facilitiate maintenance and installation. A police panel is furnished with two police-controlled switches, an integral part of the cabinet door opening with a standard police lock and key. The cabinet is floorless, permitting ease of conduit locations and making installation faster and easier. The exterior of the cabinet is finished with one coat of battleship gray and two basic coats of primer. The inside of the cabinet is painted with a lighter gray for ease of maintenance. The cabinet includes a ventilation system consisting of a 115 CFM fan on the output vent located at the top of the cabinet. The intake louvers are located on the door and are provided with a replaceable, cleanable filter. Although the cabinet is supplied with shelves in a fixed position, the shelves are adjustable on a 2 in. pitch location. The P-1 cabinet is anchored by a concrete base anchor bolt No. 9.

## Ancillary Equipment

The ancillary equipment of the Bus Detector/Intersection Control System includes all circuitry necessary to allow the BPCU to function as an interconnected intersection controller. The ancillary equipment includes solid-state load switches, flash capability, and master interconnect capability.

In addition, the cabinet includes transfer circuitry to allow the system to function with another fully independent controller as a backup unit.

When used with a second unit, the two controllers will run totally independent of each other, not sharing any components. All switching will be external and will only affect the traffic signal circuits.

Transfer of the signal circuits from the city controller to the Honeywell controller will only take place when the Honeywell controller is in the main street yellow, and the city controller has timed a minimum green. Transfer from the Honeywell controller will also take place when the Honeywell controller has completed a minimum green period and will cause the city controller to start in a main street yellow.

There are two reasons for using this transfer sequence. First, the BPCU has no built-in means of following the city controller since the transfer of control between the two is for test purposes only. In a production situation the BPCU would be a stand-alone system. Secondly, it did not seem cost effective to design and develop this capability when a much more simple scheme, i.e., one particular transfer interval, could be employed.

Unplanned transfer from the Honeywell controller back to the city controller caused by signal conflict or microprocessor failure will cause the Honeywell controller to flash for a predetermined period of time and return control to the city controller. The city controller will then resume normal operation starting in main street yellow.

Circuit Description --

Power Distribution -- As illustrated in Figures 49 and 50, cabinet power is distributed to the system through three circuit breakers. A fourth breaker controls power to the convenience outlet. The system includes lightning protection, RFI line filtering, and varistor (M. O. V.) arc suppression.

Lamp Power -- The BPCU controls the lamp drive power through individually packaged solid-state relays. Each relay is capable of supplying 10 amps. The AC+ input to the solid-state relays is controlled through two 115 vac bus transfer relays.

The first relay serves to remove power from all lamps when the signal power switch on the police panel is placed in the off position. The second relay switches lamp power from normal to flashed operation. This second relay must be held energized by the conflict monitor, the watch dog timer, and the flash switch on the police panel. If the relay is not held energized, power is removed from all signal circuits and applied to the flash circuits.

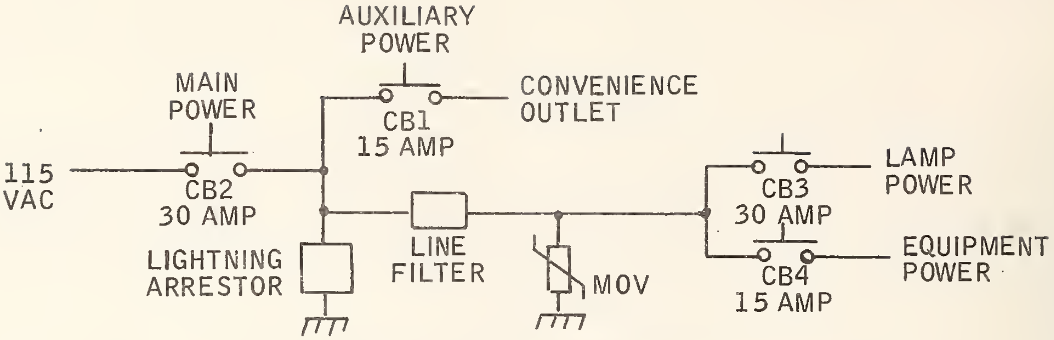


Figure 49. Power Distribution

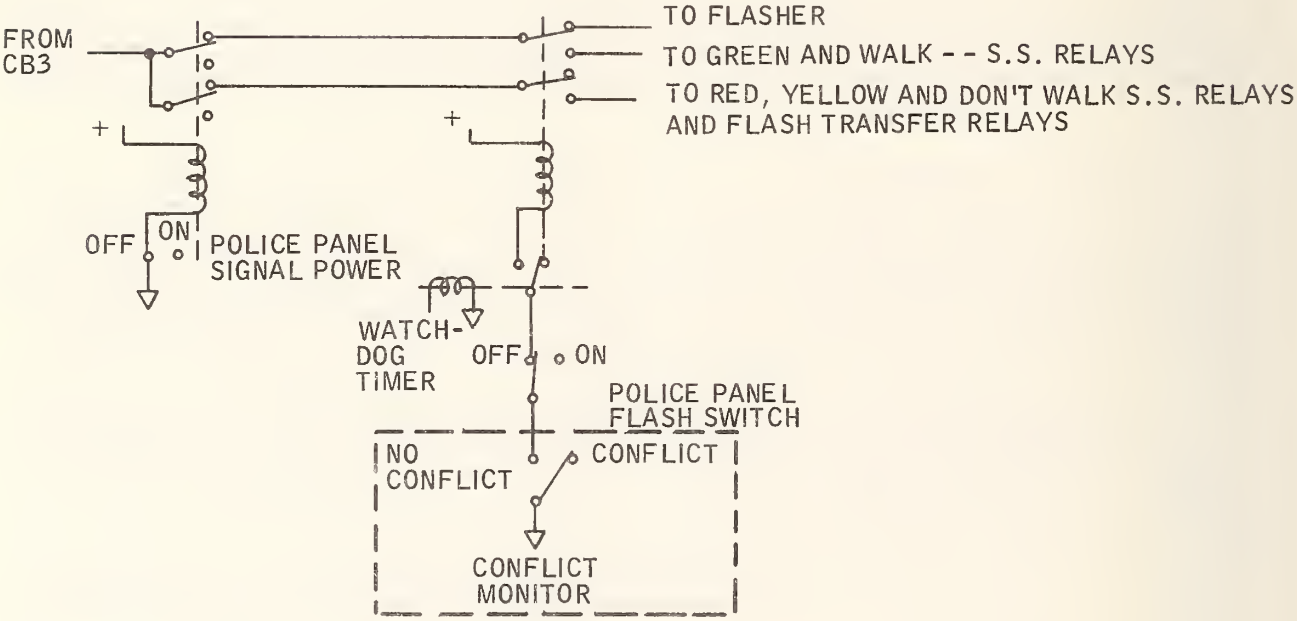

FROM CB3
TO FLASHER
TO GREEN AND WALK - - S.S. RELAYS
TO RED, YELLOW AND DON'T WALK S.S. RELAYS AND FLASH TRANSFER RELAYS
+
+
OFF
ON
POLICE PANEL SIGNAL POWER
WATCH-DOG TIMER
OFF
ON
POLICE PANEL FLASH SWITCH
NO CONFLICT
CONFLICT
CONFLICT MONITOR

Figure 50. Power Distribution (Flash Control)

Flash Operation -- The flasher unit (Figure 51) provides two 115 vac flash outputs. Each output supplies a maximum of 10 amps. The outputs pulse at a rate of 60 pulses per minute. There is a 180 deg phase shift between the two outputs. During flashing operation, flashed signal lamp connections are transferred from the solid-state relays to the flasher outputs.

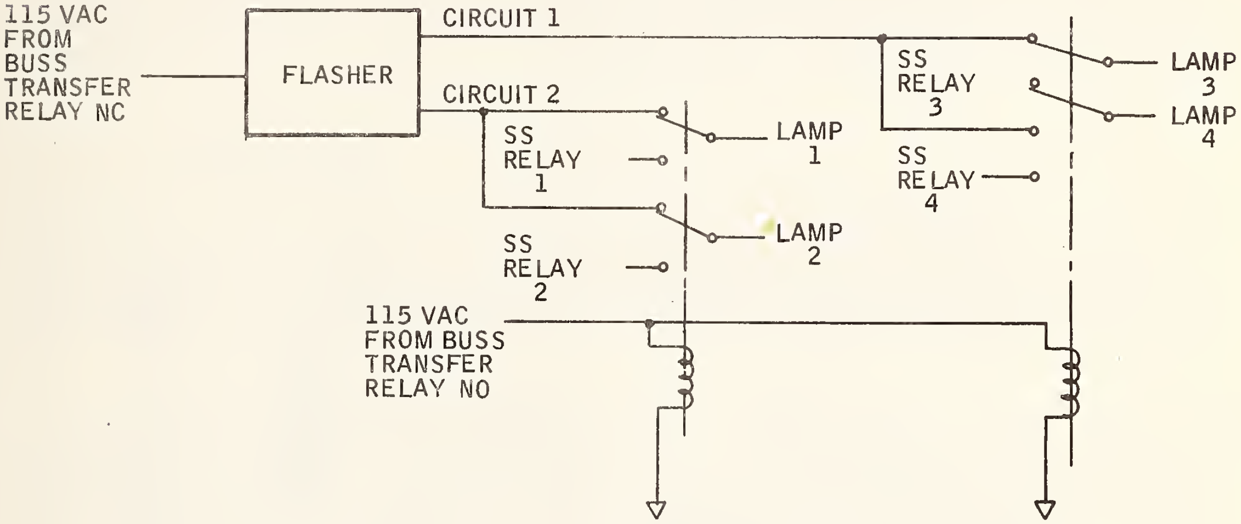


Figure 51. Flasher Circuitry

Transfer Circuitry -- As depicted in Figure 52, the transfer circuitry switches the signal lamp connections from the city controller to the BPCU. To transfer to the BPCU, the BPCU must be at the rest point of phase one yellow and the city controller must have timed a minimum green as defined by the timer T1. When the transfer button is pressed, the relay K2 is energized and then latched through K2A. If the green timer is timed out then K3 is energized through K2B. Relay K5 is energized through K3B and the transfer relays are energized through K5B. Relay K3 is latched through K4A and K3A. A scheduled request for return to city control is issued by depressing the second transfer button. Relay K2 is unlatched and when the minimum green timer times out, Relay K4 unlatches K3 and K5, thereby removing power from the transfer relays. An unscheduled transfer back to city control caused by flashing operation will result in unlatching K2, K3, and K5 at the time the "flash until transfer" timer times out.

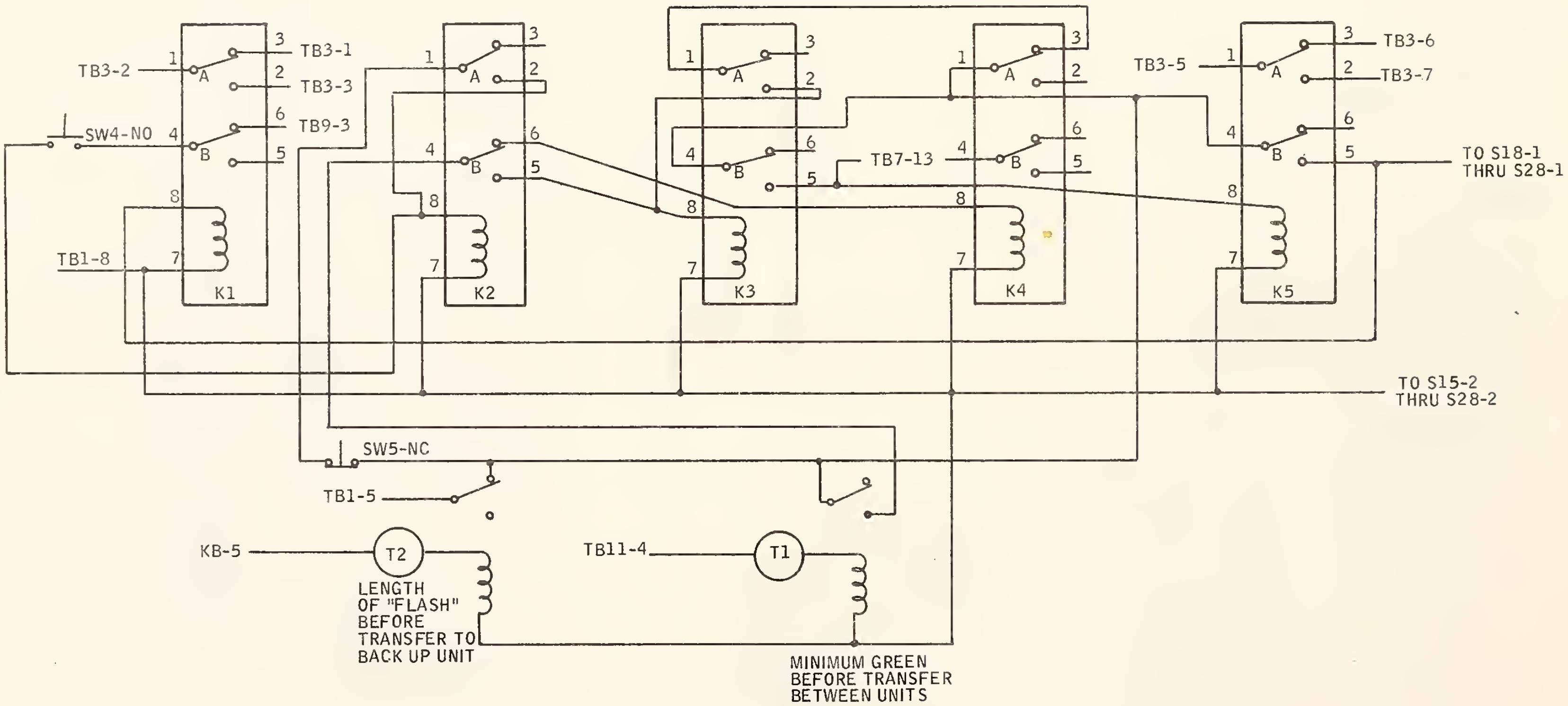


Figure 52. Transfer Circuitry

# SECTION V
# CONCLUSIONS AND RECOMMENDATIONS

## CONCLUSIONS

The Passive Bus Detector/Intersection Priority System project has been an outstanding success. All design goals have been satisfied and a new technology, initiated by this project, is emerging for advanced traffic control systems.

## RECOMMENDATIONS

As outlined in previous sections, objectives of the bus detector project have all been met successfully and feasibility of the overall concept has been demonstrated. Assuming successful field test evaluation, the next step in this project should be directed at cost-effective implementation of the passive bus detector technology.

To provide the maximum benefit of this technology to the user community and to add flexibility for implementation, it is recommended that a preemption configuration using the passive bus detector technology be developed which would interface directly with existing controllers. This preemption module would incorporate the functions of the current preprocessor, the bus classification elements of the microprocessor, a bus priority algorithm, and provide the appropriate command signals as output. Most existing electromechanical and solid-state controllers are designed to accept inputs from preemption-type equipment. Such a device could also interface with microprocessor-based controllers and with a central computer traffic control system to provide input data for traffic management control strategies.

Figure 53 is a recommended project plan for development of a Passive Bus Detector Preemptor. The critical technical issue revolves around the required speed of the microprocessors to accomplish classification. If only a few detectors must be processed, conventional microprocessors probably will be adequate. To satisfy the current objective (16 loop detectors), special-purpose, high-speed microprocessors are necessary.

A second recommendation important to implementation of passive bus detector technology concerns the bus priority algorithms. The two systems designed and developed under this contract provide the FHWA with a flexible tool to refine the existing algorithms and develop new algorithms under actual traffic conditions. It is recommended that a project be initiated to study various algorithms under different operational conditions to provide user guidelines for implementation of bus priority concepts.

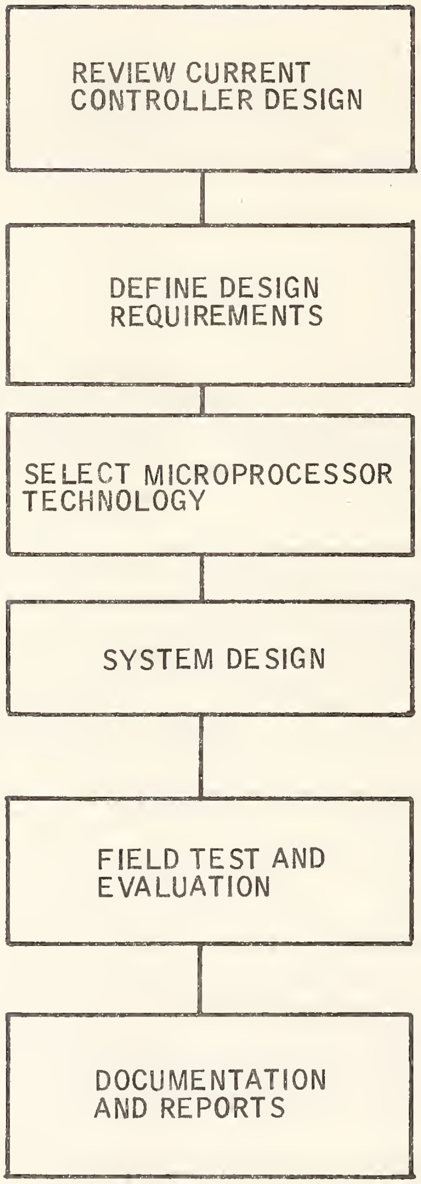

REVIEW CURRENT CONTROLLER DESIGN
DEFINE DESIGN REQUIREMENTS
SELECT MICROPROCESSOR TECHNOLOGY
SYSTEM DESIGN
FIELD TEST AND EVALUATION
DOCUMENTATION AND REPORTS

Figure 53. Passive Bus Detector Preemptor Development Plan

# APPENDIX I

## FIELD ADJUSTMENT PROCEDURE FOR BUS PRIORITY SYSTEM LOOP DETECTIONS

### PROCEDURE FOR TUNING LOOP ELECTRONICS TO LOOP AND LEAD IN

1. If loop lead-in is greater than 190 feet, set rocker switch on loop electronics board to open. If less than 190 feet, set rocker switch to closed position.

2. Set gain switch at Position 9.

3. Set the three capacitor tuning switches to 0. (Fine and coarse tuning lights should be on).

4. Increase setting on left-hand switch until coarse light goes off. Decrease one position.

5. Increase setting on middle switch until coarse light goes off. Decrease on position.

6. Increase right hand switch until coarse light and fine light goes off. Note switch setting. Increase switch position until fine light comes on. Note switch setting. Set switch at a position half way between two noted settings.

7. Monitor Pins A (+) and B (Gnd) of connector J-3 with a d-c voltmeter and readjust tuning switch for minimum output voltage. Final voltage should be $< 0.2$ volt.

   NOTE: Be sure there are no vehicles over the loop when switch positions are determined.

8. The switch-over point for capacitor range setting (190 feet) is for the loop installed in non-reinforced pavement, and loop drive frequency = 100 KHz.

9. In reinforced pavement, the switch over point (at 100 KHz) should be approximately 150 feet.

## PROCEDURE FOR SETTING LOOP ELECTRONICS GAIN

1. Nominal gain switch settings are as noted in Table II-1. The gain range toggle switch is located on the loop electronics circuit board.

TABLE II-1. NOMINAL GAIN SWITCH SETTINGS

| Loop Lead-in Length in Feet | Reinf. Pavement | | Non-Reinf. Pavement | |
|---|---|---|---|---|
| | Range | Sw. Pos. | Range | Sw. Pos. |
| 0-40 | Low | 2 | Low | 0 |
| 40-90 | | 7 | | 1 |
| 90-140 | High | 0 | | 2 |
| 140-190 | | 1 | | 2 |
| 190-240 | | 2 | | 3 |
| 240-290 | | 3 | | 4 |
| 290-340 | | 3 | | 6 |
| 340-390 | | 4 | | 7 |
| 390-440 | | 4 | | 8 |

2. Monitor Pins A (+) and B (Gnd) of Connector J-3 with a d-c voltage measuring device (chart recorder preferable). Outputs from vehicles going over the loop will vary from 2 or 3 volts to 11.5 volts (vehicles with lower ground clearance usually give higher outputs). Minimum peak outputs from buses should be 3 volts. Maximum peak output from busses should be 11 volts.

3. The gain settings and switchover points for capacitor range are not exact. The values given are nominal, based on limited data and serve only as starting points for tuning at various locations and oscillator frequencies.

# APPENDIX II

# ESTIMATION OF VEHICLE SPEED FROM BUS DETECTOR SIGNATURE DATA

## SPEED ESTIMATION

### Summary

Two ways to estimate vehicle speed were studied during the contract. The first method, called the "classifier estimate", relies on the classifier to identify the vehicle; then the speed is calculated from the known vehicle length and the observed time. The second method, called the "MGVD slope estimate" uses the slope of the MGVD phase response signal (which theoretically depends only upon the speed of the vehicle) to estimate the speed. Both methods were successful.

The classifier estimate has the obvious disadvantage that it can only provide speed estimates for the vehicles that are uniquely classified -- in this case, the buses. Even then the method requires an accurate vehicle length for each class. In this case, the buses of interest all had essentially the same length, but if mini and articulated buses are included the classifier needs to be more complex. As shown in the following detailed discussion, the speed estimates are accurate to ±10 percent.

The MGVD slope estimate does not depend upon knowledge of vehicle type and so has a distinct advantage as a speed estimation method. Only 17 vehicle runs were useable for this analysis, but they provide strong experimental evidence for the slope being dependent only upon the speed. The speed estimates are again about ±10 percent.

## CLASSIFIER ESTIMATE

This method assumes that the first and last peaks of a bus signature are produced by the front and rear axles of the bus. The total time T between the first peak and the last peak of a bus signal is measured as a primary feature and can be used to estimate the bus' speed.

If we assume

- constant speed
- first and last peaks mark axles
- bus wheel base = 23 ft. 8 in.

Then

$$\text{Speed} = \frac{23.67}{T} \cdot \frac{30}{44} = \frac{16.14}{T} \text{ mph}$$

where

T is in seconds.

This calculation was compared with the speed estimate obtained from the presence detector (interrupt light beam) output that was recorded during the vehicle runs. It was found that the peak-to-peak estimate was in error by a constant factor of 1/1.3. (This is probably due to the last peak being caused by the combined effects of the rear axle and the rear mounted engine.) The corrected formula which is implemented in the simulation is thus:

$$S_2 = \text{Speed (corrected)} = \frac{20.98}{T} \text{ mph}$$

The graph ($S_2$ vs $S_1$), Figure II-1, shows the calculated speed $S_2$ (peak-to-peak corrected) and the measured speed $S_1$ (presence detector) for each of the runs on asphalt. The scatter about the line $S_1 = S_2$ is attributed to acceleration or deceleration while the bus was crossing the loop, plus errors in reading the total time from the presence detector marker on the strip chart recordings.

## MGVD SLOPE ESTIMATE

Fifty-five experimental runs were made to test the phase-optimized MGVD. This was a multi-purpose test involving four vehicles (VW, step van, station wagon and bus) and recording signals from both the MGVD and the magnetic loop. Unfortunately, the presence detector stopped working after the 24th run. Since the presence detector provides the speed measurement (nominal speeds are very inaccurate), this gave a very small data base for testing the speed estimation hypothesis.

The useable runs (Table II-1) were made by the VW (nine runs) and the step van (eight runs). The step van is about 8 feet tall with flat, vertical surfaces at the front and rear. The VW was a standard bug. Thus, the two vehicles were very dissimilar, particularly in their front and rear surfaces where vehicle differences might affect the MGVD signal slope.

The slopes were estimated by measuring the linear portion of the signal trace on a Brush recording made during the experimental run. The speeds were determined by using the presence detector mark on the Brush recording to indicate the time, and using the nominal lengths of 13.25 feet for the VW and 23.75 feet for the step van. Slope estimates were made at both the front and back parts of each signal. The measurements are given in Table II-1.

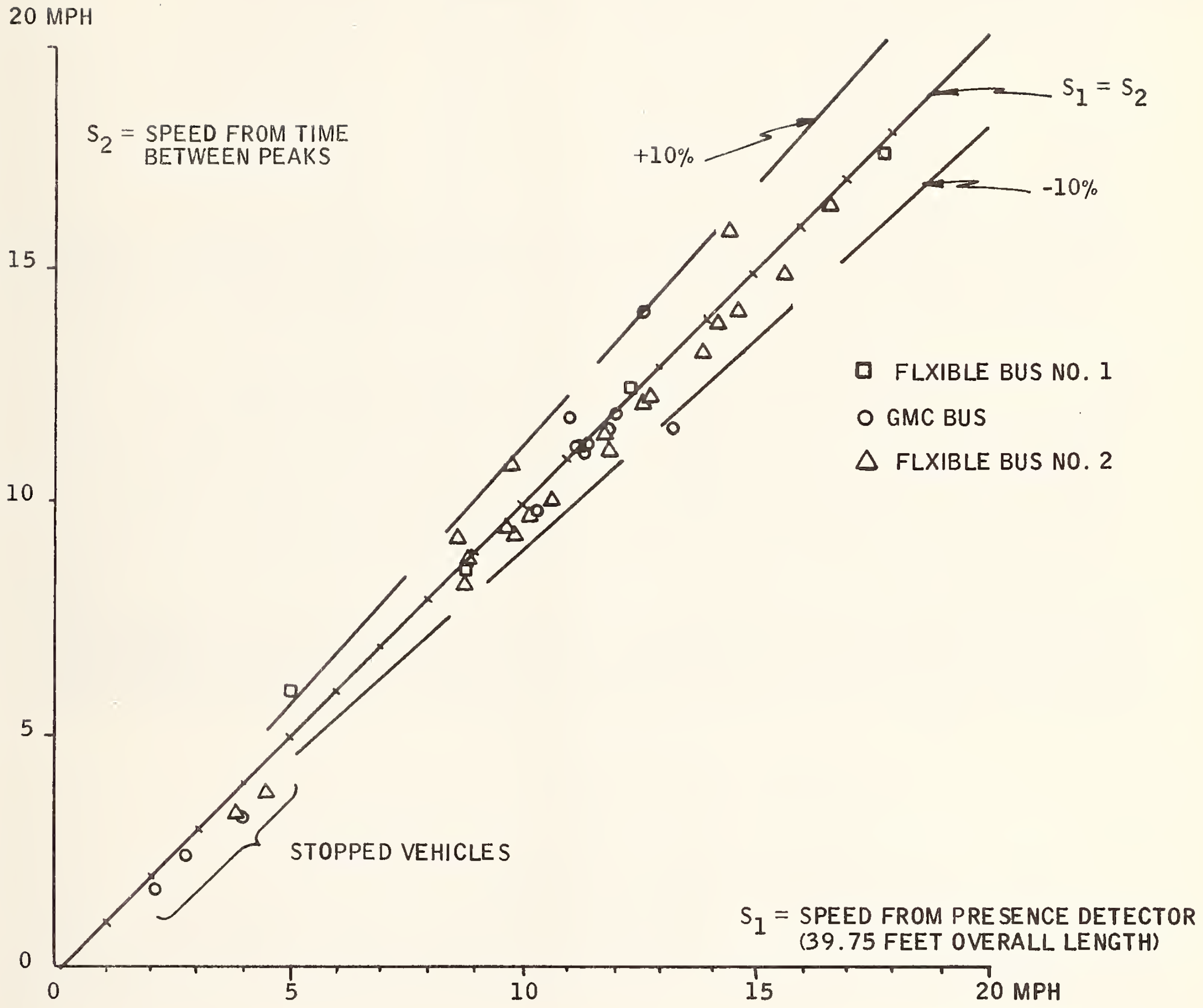


Figure II-1. Calculated Speed vs Measured Speed: Classifier Estimate

TABLE II-1

| Vehicle | Run | (ft/sec) | Front Slope | Rear Slope |
|---|---|---|---|---|
| VW | 395 | 14.6 | 3.1 | 3.5 |
| | 396 | 14.2 | 3.1 | 3.5 |
| | 397 | 13.6 | 3.0 | 3.2 |
| | 398 | 13.2 | 2.9 | 2.9 |
| | 401 | 13.6 | 2.7 | 3.1 |
| | 402 | 13.2 | 2.5 | 2.9 |
| | 402a | 13.6 | 2.8 | 3.0 |
| | 405 | 9.0 | 1.9 | 2.3 |
| | 406 | 22.7 | 4.8 | 5.2 |
| Step Van | 407 | 12.7 | 2.6 | 3.5 |
| | 408 | 13.2 | 2.5 | 3.4 |
| | 409 | 12.0 | 2.3 | 2.8 |
| | 410 | 14.1 | 2.5 | 3.3 |
| | 413 | 15.1 | 3.1 | 2.7 |
| | 414 | 12.8 | 2.4 | 2.5 |
| | 417 | 8.8 | 1.6 | 2.5 |
| | 418 | 23.3 | 4.4 | 5.8 |

The model we want to test is that the speed V is proportional to the slope S in a way that does not depend on the vehicle. Thus, we want to determine the proportionality constant k so that V = kS and test whether k depends on the vehicle. The data cable contains four separate cases that can be compared:

| | |
|---|---|
| Case 1 | VW, front |
| Case 2 | VW, rear |
| Case 3 | Step van, front |
| Case 4 | Step van, Rear |

The least-square estimator of k that minimizes $\sum_i (V_i - k\,S_i)^2$ is given by

$$k = \frac{\sum_1 V_i S_i}{\sum_i S_i^2}$$

For these cases, we get

| Case | k |
|---|---|
| 1 | 4.75 |
| 2 | 4.32 |
| 3 | 5.22 |
| 4 | 4.16 |

At first glance, this appears inconclusive. The four estimates are quite different, but then the scatter in the slope values is quite large and this may account for the differences. A closer look at the table reveals what might be a systematic error in the data.

The rear slopes are consistently larger than the front slopes (with one exception). This does not appear to be a random effect introduced during the slope measurement. In searching for an experimental bias that might cause these differences, one is led to the possibility that these two vehicles were both accelerating as they crossed the MGVD. This is a reasonable possibility; furthermore, such being the case would cause the speed increase (and therefore the slope increase) to be larger for the longer vehicle -- and this is the case.

Operating under this assumption, the estimate should be free of this effect if the average of the front and rear slopes is used. Doing this yields

$$k_{VW} = 4.53$$

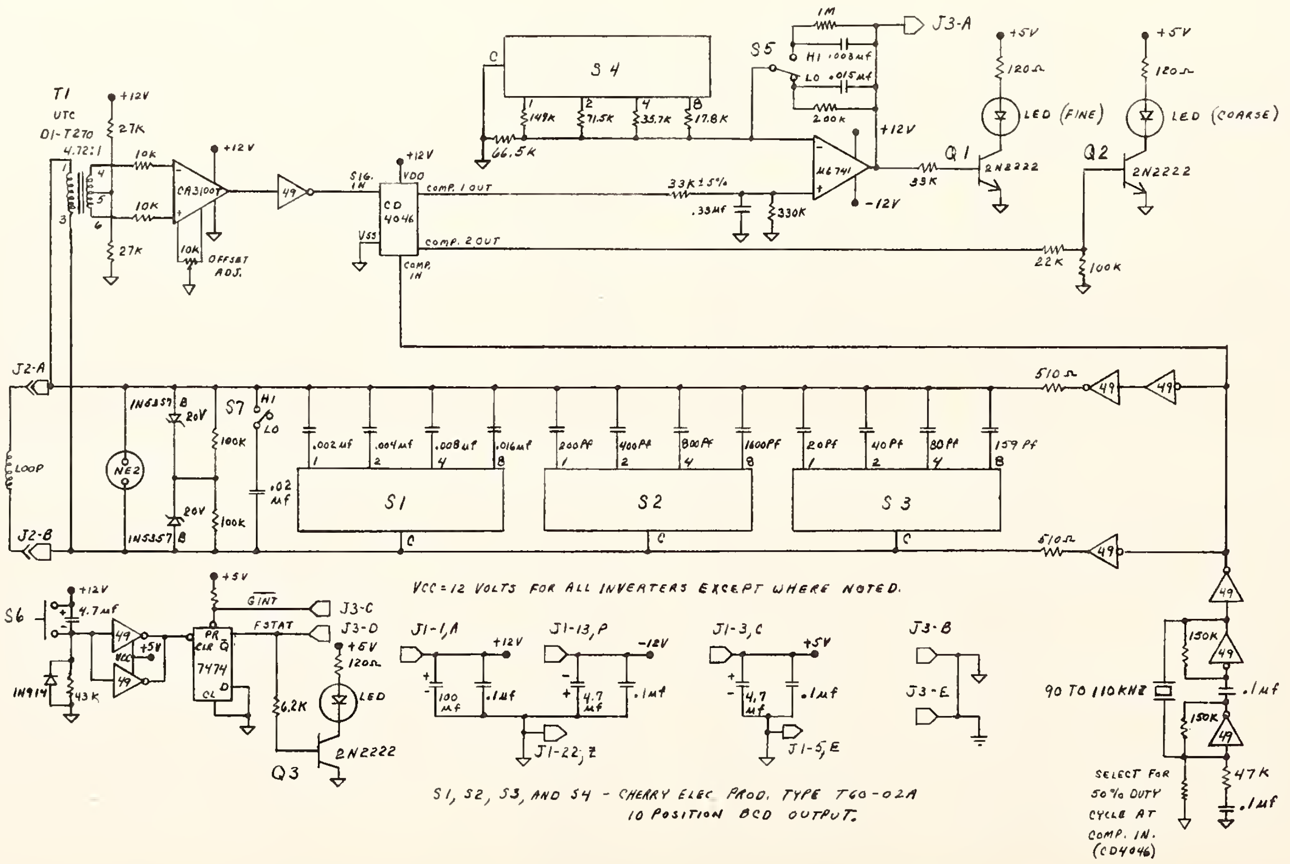
$$k_{\text{step van}} = 4.66$$

The difference in these values is so small that we can easily conclude the difference between the vehicles is insignificant.

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



VCC=12 VOLTS FOR ALL INVERTERS EXCEPT WHERE NOTED.

S1, S2, S3, AND S4 - CHERRY ELEC. PROD. TYPE T60-02A 10 POSITION BCD OUTPUT.

SELECT FOR 50% DUTY CYCLE AT COMP. IN. (CD4046)

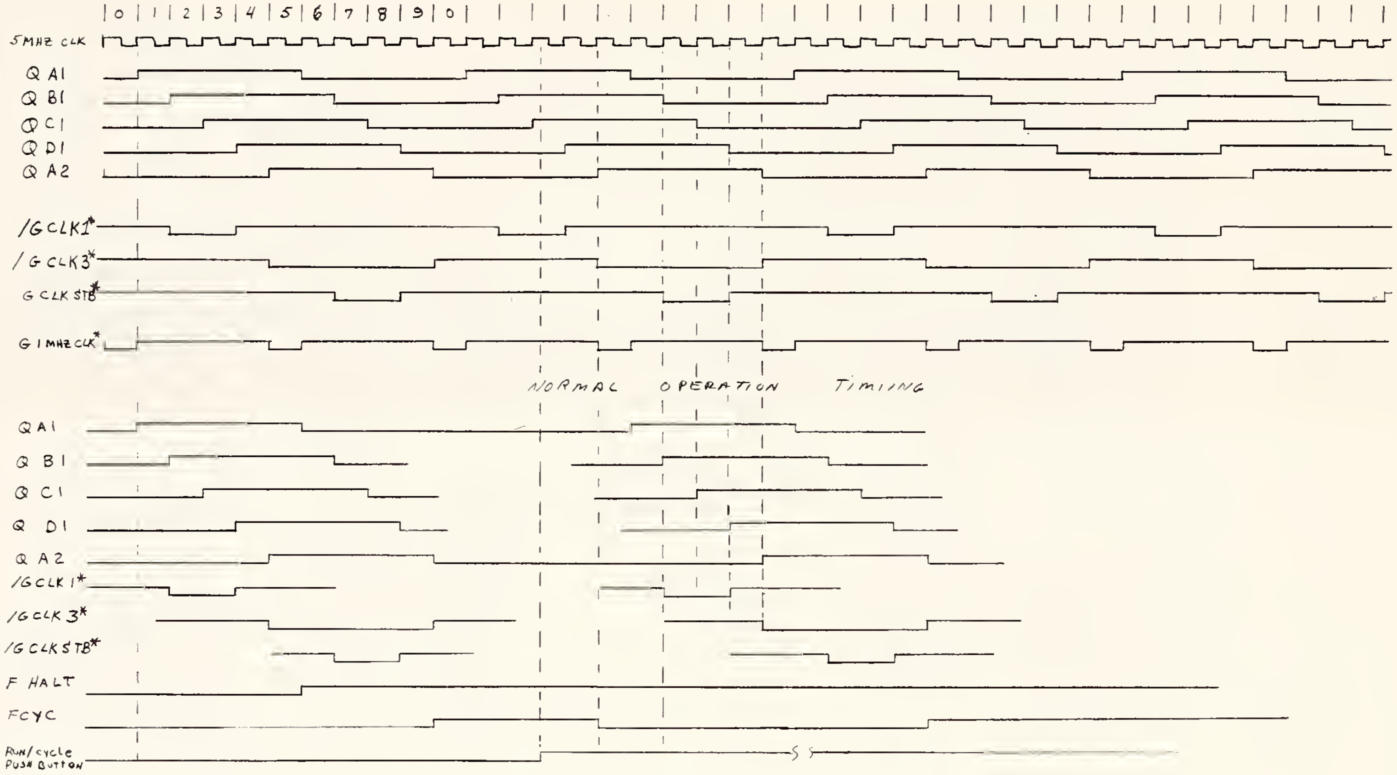
| | | TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° ±1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|
| | | MATERIAL | CONTRACT NO | LOOP ELECTRONICS | |
| NEXT ASSY | USED ON | | | SIZE C | CODE IDENT NO. 27327 / DRAWING NO. |
| APPLICATION | | FINISH—SEE NOTE | | SCALE | SHEET 1 OF 1 |

DF-8 REV. 7-79 V000

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



| | | | |
|---|---|---|---|
| TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ± | 90° FORMED ANGLES +2° -1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | |
| MATERIAL | | CONTRACT NO | |
| NEXT ASSY | USED ON | | |
| APPLICATION | | FINISH–SEE NOTE | |

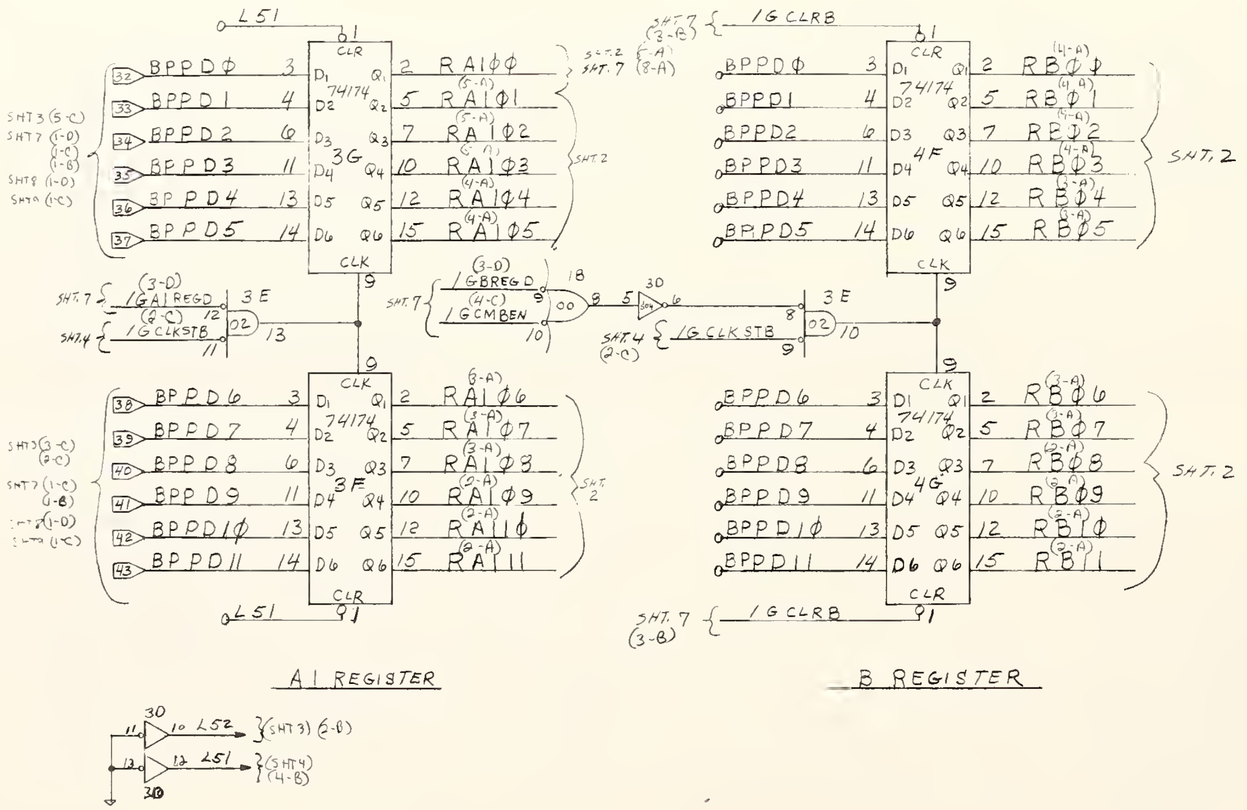
| | |
|---|---|
| HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
| BUS DETECTOR P.P.<br>BASIC TIMING | |
| SIZE C | CODE IDENT NO. 27327 | DRAWING NO. |
| SCALE | SHEET 1 OF 1 |

DF-8 REV. 7-75 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-O-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |

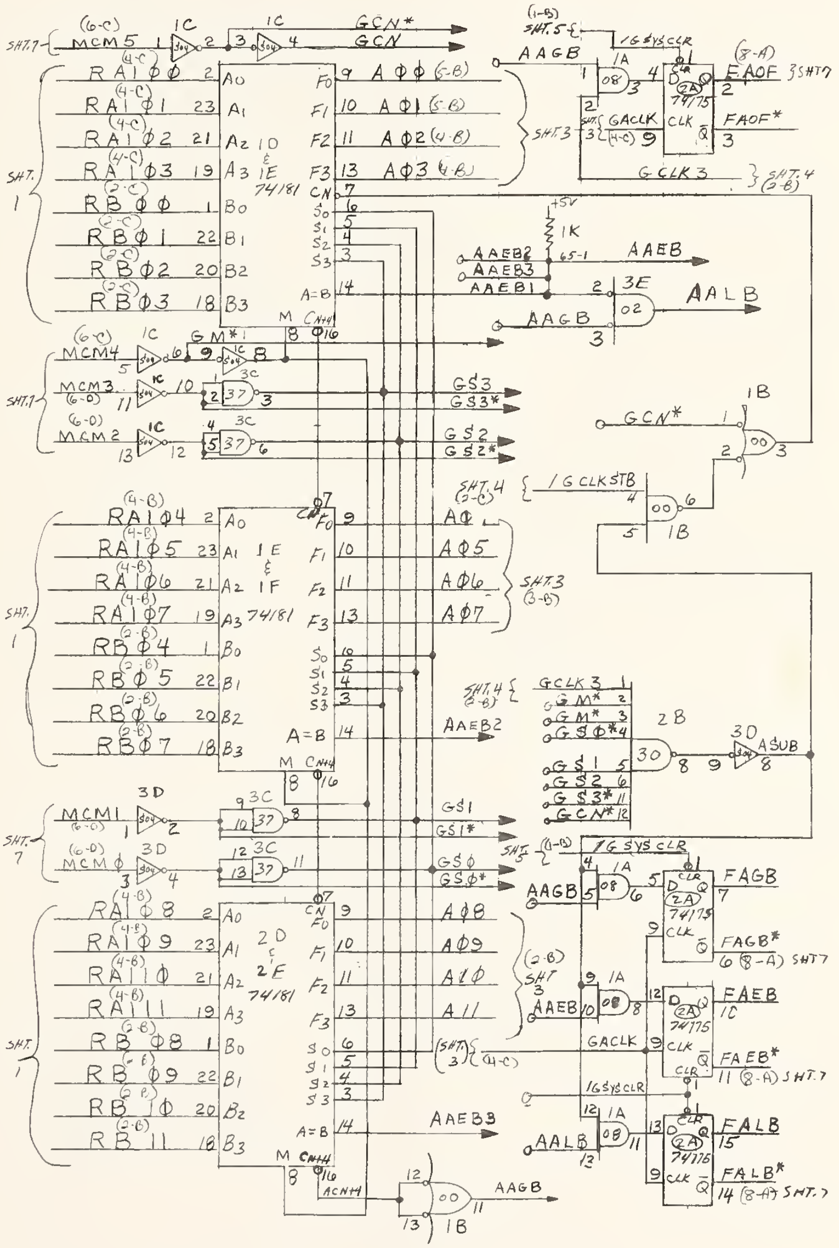


| | | TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° ±1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|
| | | MATERIAL | CONTRACT NO. | BUS DETECTOR<br>PRE-PROCESSOR<br>A/B REGISTERS | |
| NEXT ASSY | USED ON | | | SIZE<br>C | CODE IDENT NO.<br>27327 | DRAWING NO. |
| APPLICATION | | FINISH–SEE NOTE | | SCALE | SHEET 1 OF 9 |

DF-8 REV. 7-79 V008

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|



CHANGED 6/3/75 (AADD + ASUB)
CHANGED 6/10/75 (ASUB DELETE AADD)

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

TOLERANCES UNLESS NOTED OTHERWISE
X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2°
±.10

MATERIAL

FINISH—SEE NOTE

NEXT ASSY | USED ON
APPLICATION

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT

CONTRACT NO

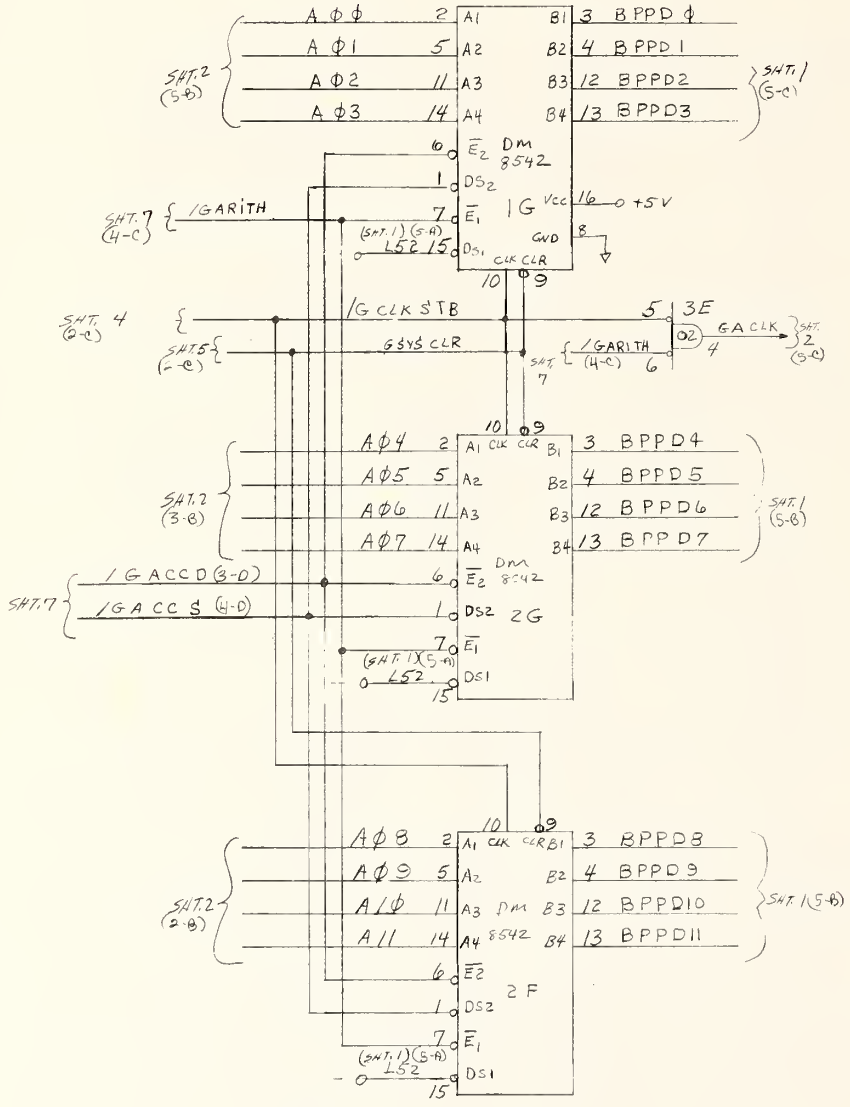
HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

BUS DETECTOR
P.P. A.L.U.

SIZE C | CODE IDENT NO. 27327 | DRAWING NO.

SCALE | SHEET 2 OF 9

DF-5 REV. 7-75 V600



CARD #1

HONEYWELL INC.

SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

BUS DETECTOR
P.P. ACCUM REG-CARD #1

| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| C | 27327 | |

SCALE

SHEET 3 OF 9

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2° ±1°

MATERIAL

FINISH-SEE NOTE

NEXT ASSY | USED ON

APPLICATION

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

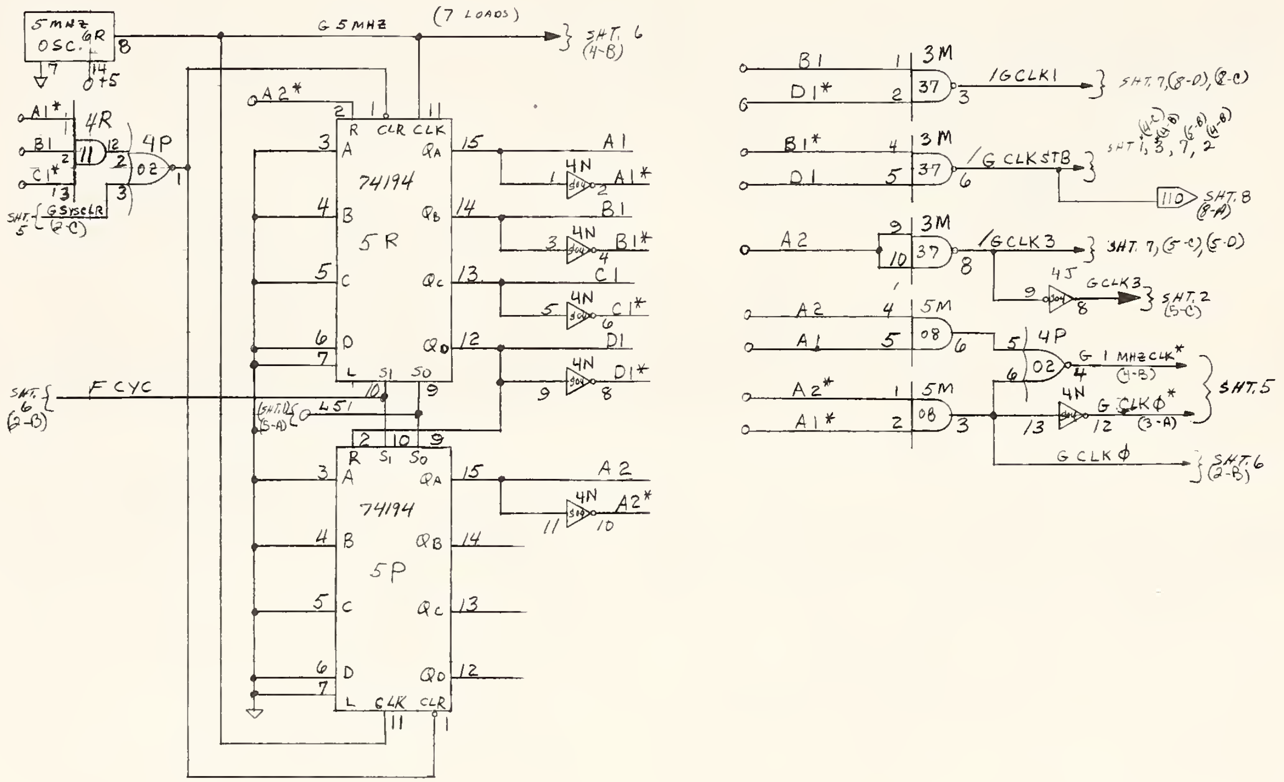
HONEYWELL PART NO.

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

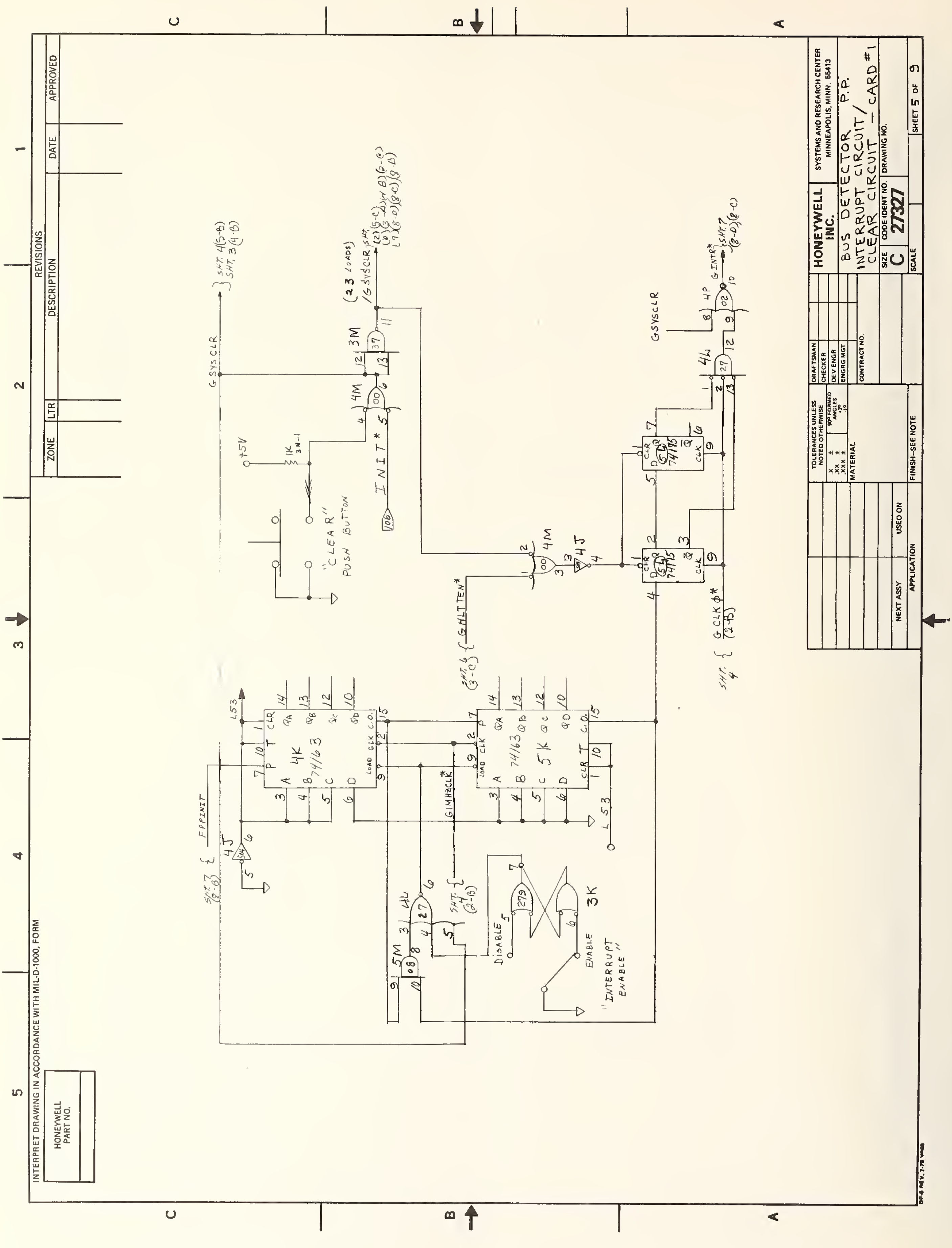
REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |



| | | TOLERANCES UNLESS NOTED OTHERWISE | DRAFTSMAN | | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|---|
| | | .X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° 1° | CHECKER<br>DEV ENGR<br>ENGRG MGT | | BUS DETECTOR P.P.<br>CLK CIRUIT - CARD #1 | |
| | | MATERIAL | CONTRACT NO. | | SIZE C | CODE IDENT NO. 27327 / DRAWING NO. |
| NEXT ASSY | USED ON | | | | | |
| APPLICATION | | FINISH–SEE NOTE | | | SCALE | SHEET 4 OF 9 |

DF-8 REV. 7-75 V600

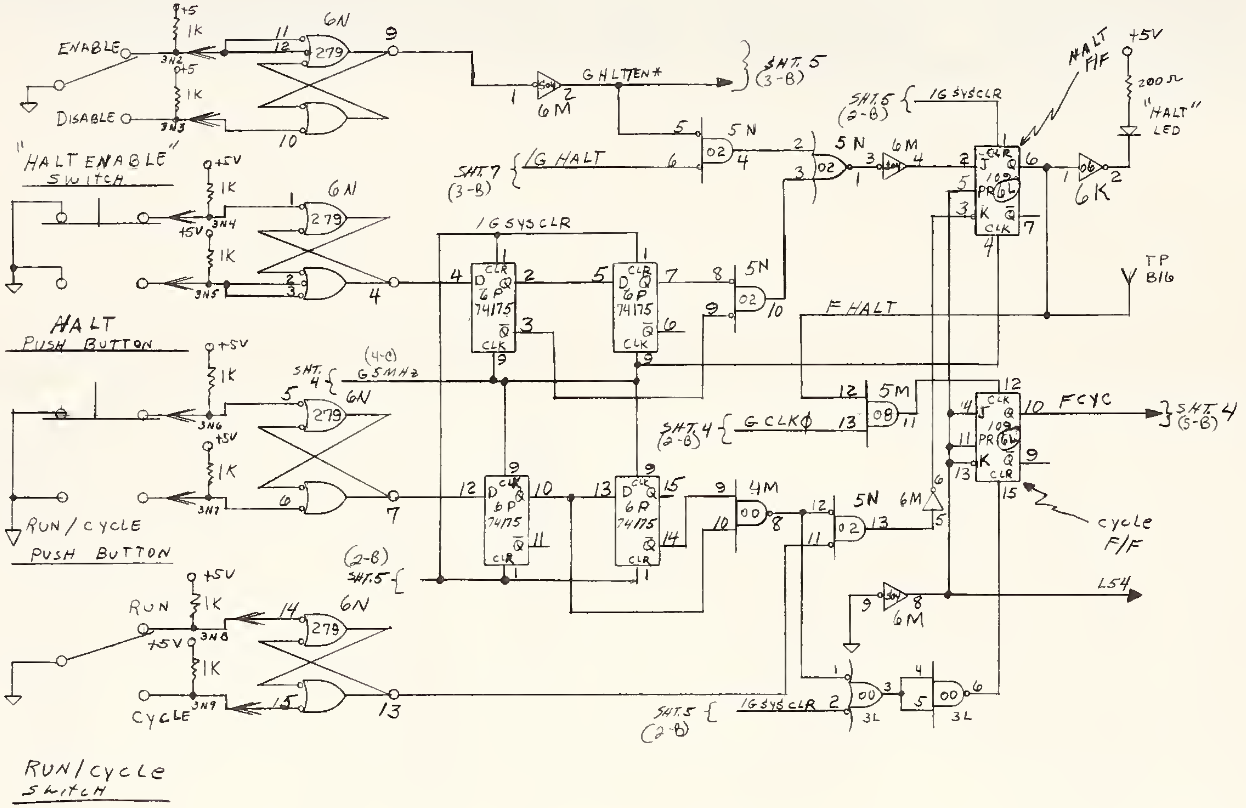

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM
HONEYWELL PART NO.
REVISIONS
ZONE
LTR
DESCRIPTION
DATE
APPROVED
HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413
BUS DETECTOR P.P. INTERRUPT CIRCUIT/ CLEAR CIRCUIT — CARD #1
SIZE C
CODE IDENT NO. 27327
DRAWING NO.
SCALE
SHEET 5 OF 9
TOLERANCES UNLESS NOTED OTHERWISE
MATERIAL
FINISH–SEE NOTE
DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.
NEXT ASSY
USED ON
APPLICATION
G SYS CLR
SHT. 4(5-B) SHT. 3(4-B)
(23 LOADS) /G SYSCLR
+5V
1K 3N-1
"CLEAR" PUSH BUTTON
INIT*
4M
3M
G SYSCLR
4L
4P
G INTR*
SHT. 7 (8-D)(8-C)
GHLTTEN*
SHT. 6 (3-C)
4J
5L
74175
GCLKφ*
SHT. 4 (2-B)
4K
74/163
5K
74/163
L53
FPPINIT
SHT. 7 (8-B)
GIMHZCLK*
SHT. 4 (2-B)
5M
DISABLE
ENABLE
"INTERRUPT ENABLE"
3K
279

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |



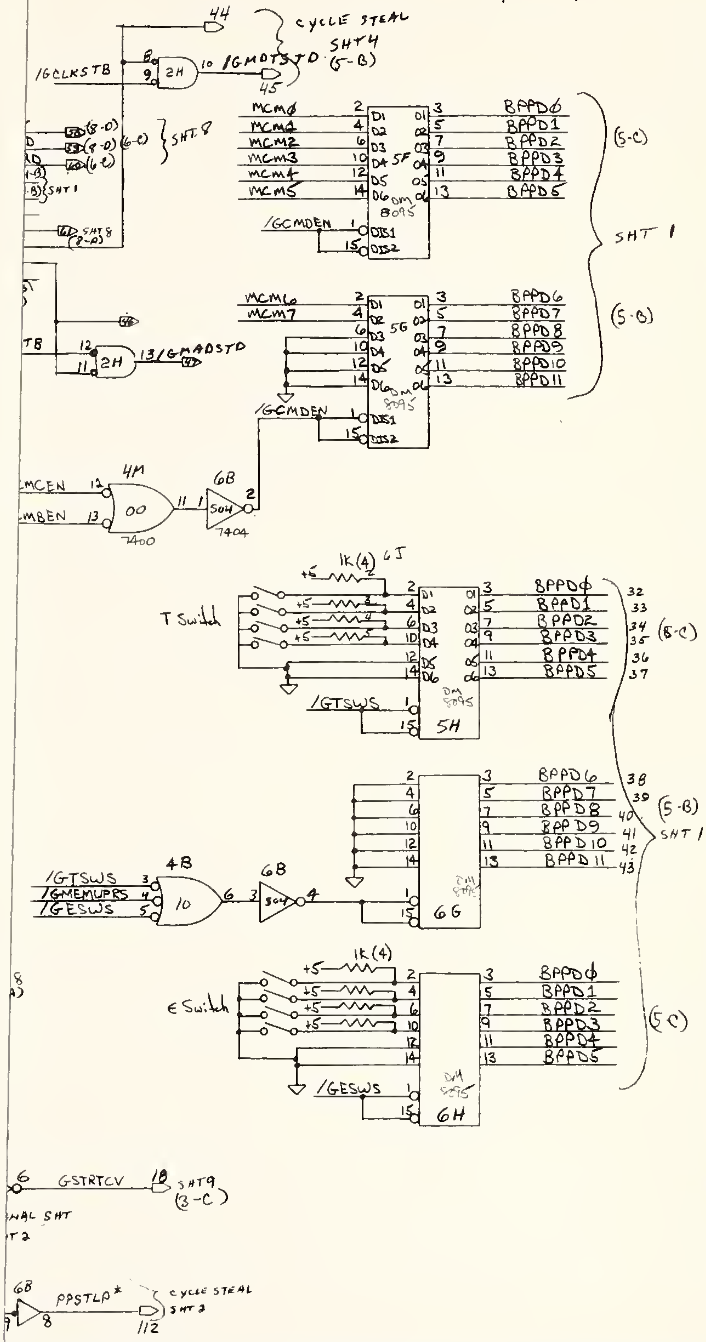
| | | TOLERANCES UNLESS NOTED OTHERWISE | DRAFTSMAN | | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|---|
| | | .X ± <br> .XX ± <br> .XXX ± <br> 90° FORMED ANGLES ±2° ±1° | CHECKER | | BUS DETECTOR P.P. <br> HALT/CYCLE <br> CONTROL CIRCUIT - CARD #1 | |
| | | | DEV ENGR | | | |
| | | | ENGRG MGT | | | |
| | | MATERIAL | CONTRACT NO. | | SIZE C | CODE IDENT NO. 27327 | DRAWING NO. |
| NEXT ASSY | USED ON | | | | | |
| APPLICATION | | FINISH–SEE NOTE | | | SCALE | SHEET 6 OF 9 |

DF-5 REV. 7-75 V600

2 1

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |

D



C

B

A

| | | | |
|---|---|---|---|
| SS<br>E | DRAFTSMAN | | |
| | CHECK | | |
| RMED<br>LES<br>2°<br>1° | DEV ENGR | | |
| | PROJ ENGR | | |
| | RELIABILITY | | |
| | | | |
| | CONTRACT NO. | | |

HONEYWELL INC. AEROSPACE AND DEFENSE GROUP
□ AERD MPLS □ AERD ST PETERSBURG □ DRO SEATTLE □ DRO HDPKINS
□ DRO MONTGOMERYVILLE □ S&R CTR MPLS □ RAOIATION CTR BDSTDN

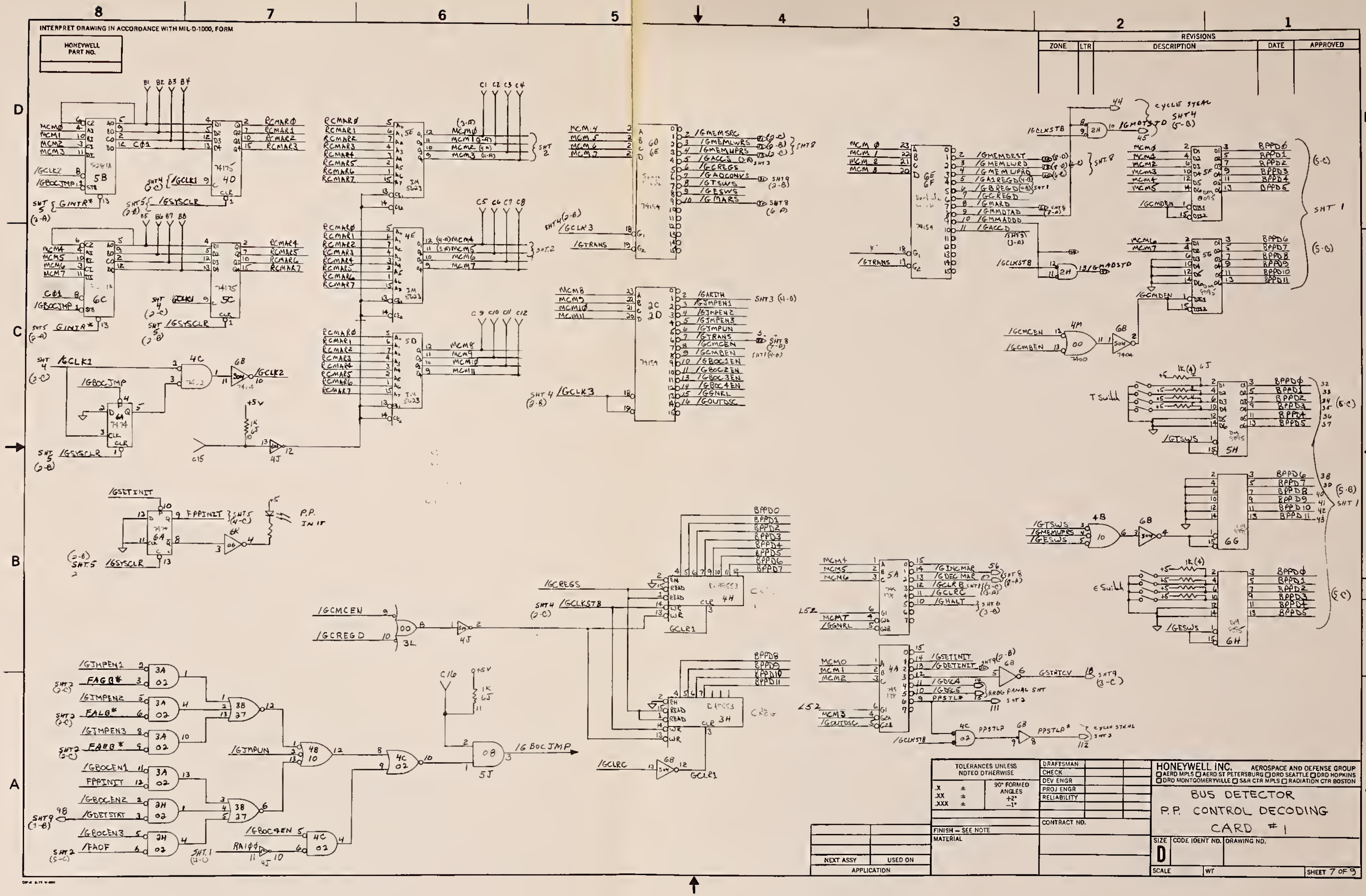
BUS DETECTOR
P.P. CONTROL DECODING
CARD #1

| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| D | | |

| SCALE | WT | SHEET 7 OF 9 |
|---|---|---|

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.



| ZONE | LTR | REVISIONS DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES +2° −1°

DRAFTSMAN
CHECK
DEV ENGR
PROJ ENGR
RELIABILITY
CONTRACT NO.
FINISH – SEE NOTE
MATERIAL
NEXT ASSY
USED ON
APPLICATION

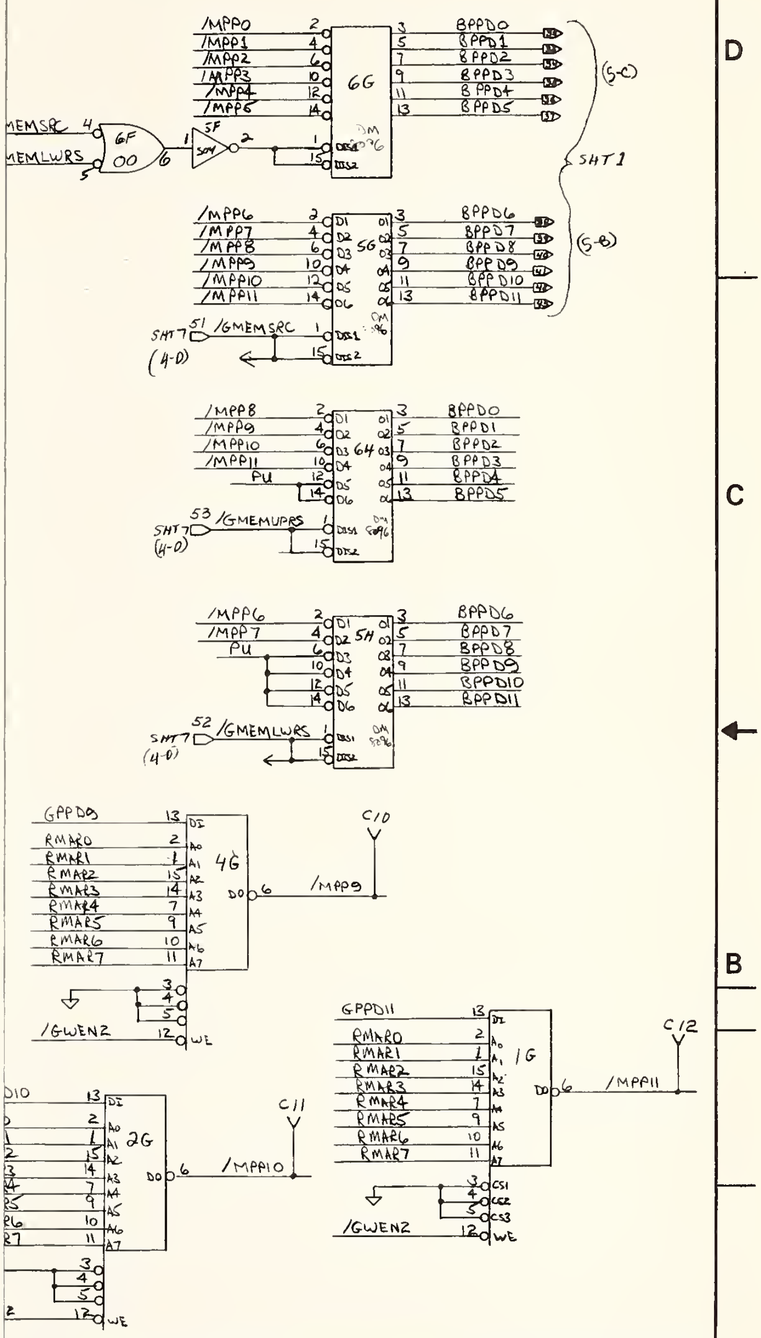
HONEYWELL INC. AEROSPACE AND DEFENSE GROUP
□ AERO MPLS □ AERO ST PETERSBURG □ DRO SEATTLE □ DRO HOPKINS □ DRO MONTGOMERYVILLE □ S&R CTR MPLS □ RADIATION CTR BOSTON

BUS DETECTOR
P.P. CONTROL DECODING
CARD #1

SIZE D
CODE IDENT NO.
DRAWING NO.
SCALE
WT
SHEET 7 OF 9

2 1

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



| | | |
|---|---|---|
| SS<br>E | DRAFTSMAN | |
| | CHECK | |
| ORMEO<br>GLES<br>±2°<br>±1° | OEV ENGR | |
| | PROJ ENGR | |
| | RELIABILITY | |
| | CONTRACT NO. | |

HONEYWELL INC. AEROSPACE AND DEFENSE GROUP

□ AERO MPLS □ AERO ST PETERSBURG □ ORO SEATTLE □ ORO HOPKINS
□ ORO MONTGOMERYVILLE □ S&R CTR MPLS □ RAOIATION CTR BOSTON

BUS DETECTOR P.P.

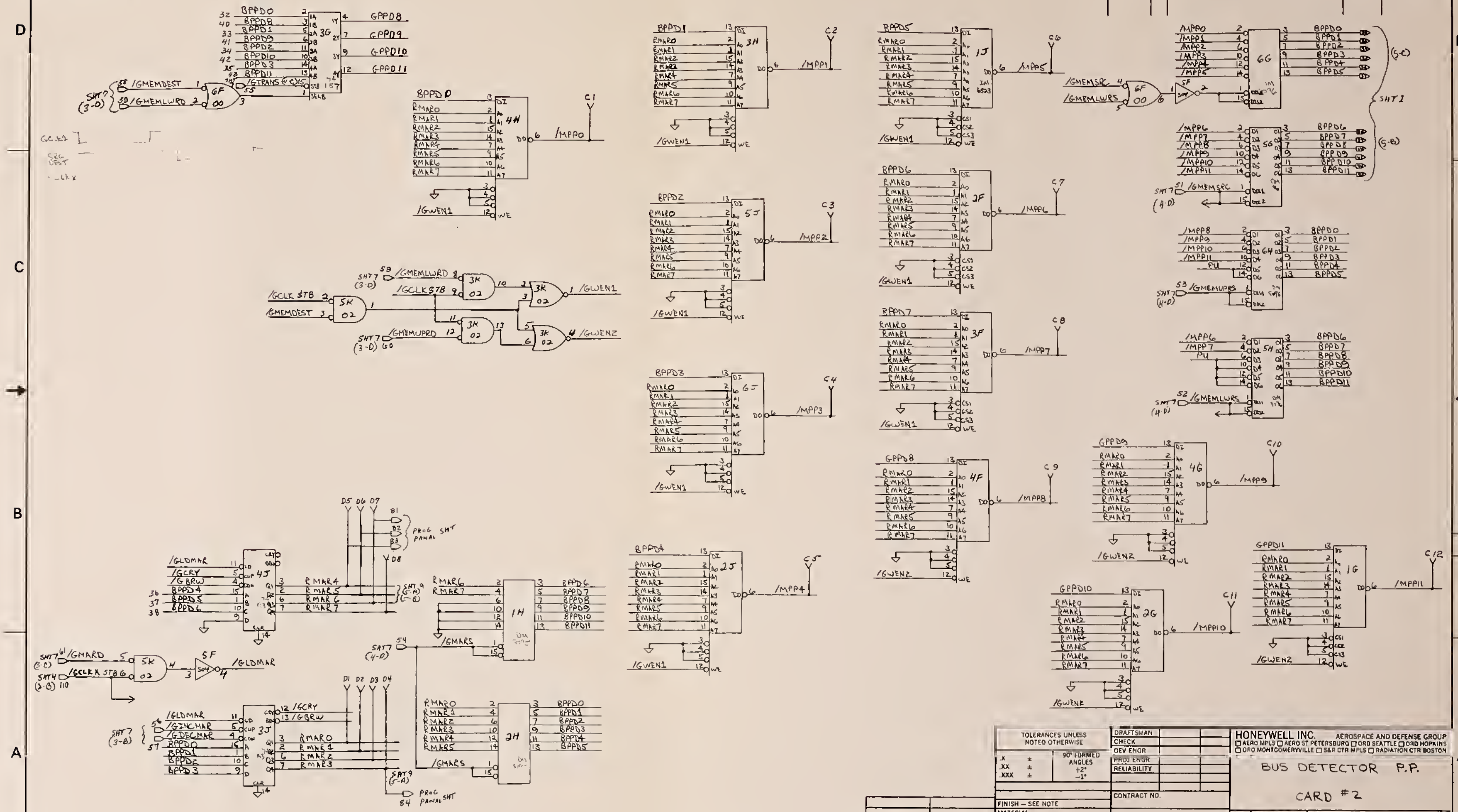
CARD #2

| SIZE | COOE IOENT NO. | DRAWING NO. |
|---|---|---|
| D | 27327 | |

SCALE | WT | SHEET 8 OF 9

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.



| | | |
|---|---|---|
| TOLERANCES UNLESS NOTED OTHERWISE | DRAFTSMAN | |
| .X ± | CHECK | |
| .XX ± | DEV ENGR | |
| .XXX ± | PROJ ENGR | |
| 90° FORMED ANGLES +2° −1° | RELIABILITY | |
| FINISH — SEE NOTE | CONTRACT NO. | |
| MATERIAL | | |

| NEXT ASSY | USED ON |
|---|---|
| APPLICATION | |

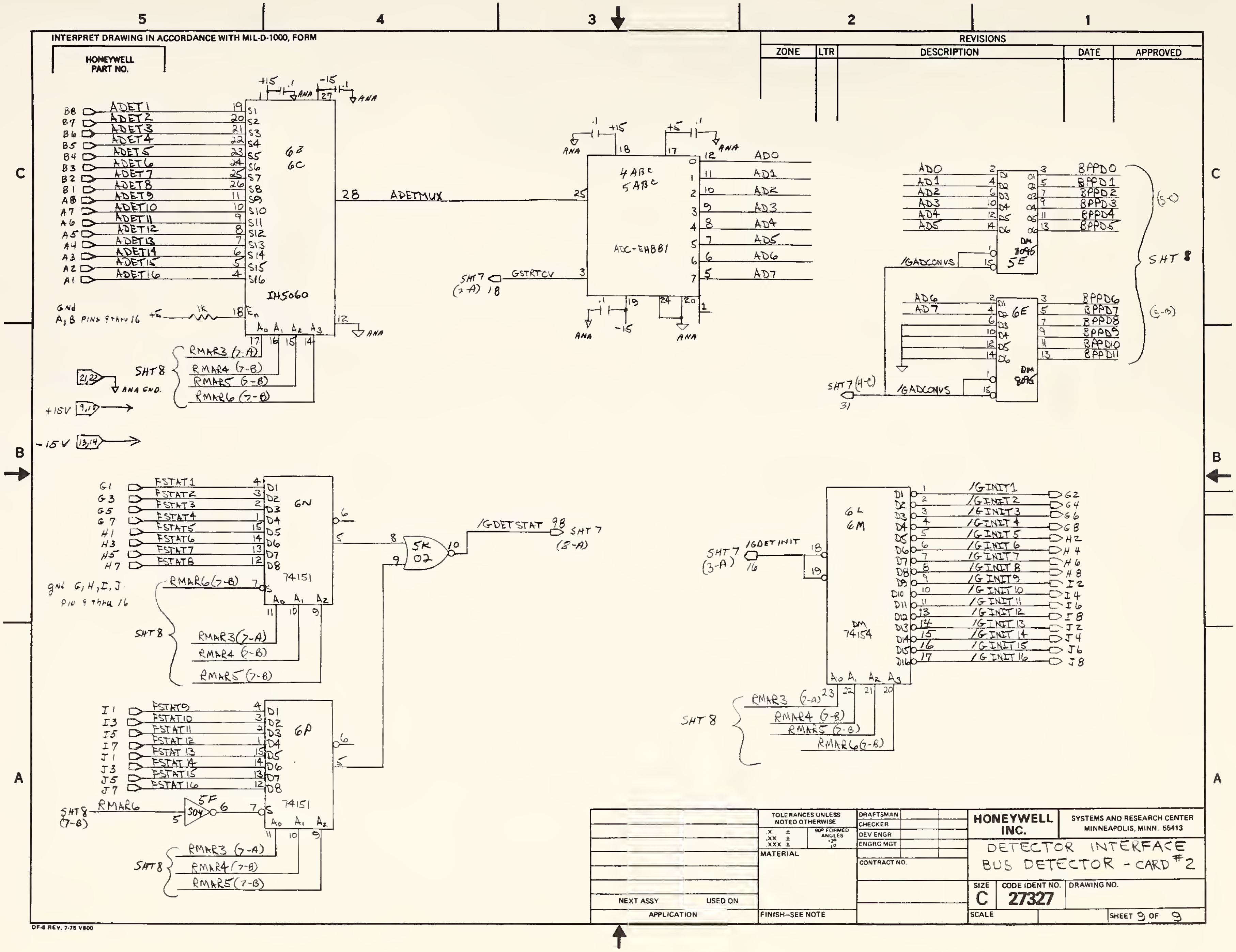
HONEYWELL INC. AEROSPACE AND DEFENSE GROUP
□ AERO MPLS □ AERO ST PETERSBURG □ ORD SEATTLE □ ORD HOPKINS
□ ORD MONTGOMERYVILLE □ S&R CTR MPLS □ RADIATION CTR BOSTON

BUS DETECTOR P.P.

CARD #2

| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| D | 27327 | |

| SCALE | WT | SHEET 8 OF 9 |
|---|---|---|


INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM
HONEYWELL PART NO.
REVISIONS
ZONE
LTR
DESCRIPTION
DATE
APPROVED
B8 B7 B6 B5 B4 B3 B2 B1 A8 A7 A6 A5 A4 A3 A2 A1
ADET1 ADET2 ADET3 ADET4 ADET5 ADET6 ADET7 ADET8 ADET9 ADET10 ADET11 ADET12 ADET13 ADET14 ADET15 ADET16
6B
6C
IH5060
28
ADETMUX
GND A, B PINS 9 thru 16
+5
1K
RMAR3 (7-A)
RMAR4 (7-B)
RMAR5 (6-B)
RMAR6 (7-B)
SHT8
21,22
ANA GND.
+15V 9,10
-15V 13,14
ANA
4ABC
5ABC
ADC-EH8B1
SHT7 (2-A) 18
GSTRTCV
AD0 AD1 AD2 AD3 AD4 AD5 AD6 AD7
DM 8095
5E
6E
BPPD0 BPPD1 BPPD2 BPPD3 BPPD4 BPPD5
BPPD6 BPPD7 BPPD8 BPPD9 BPPD10 BPPD11
(5-C)
(5-B)
SHT 8
/GADCONVS
SHT7 (4-C) 31
G1 G3 G5 G7 H1 H3 H5 H7
FSTAT1 FSTAT2 FSTAT3 FSTAT4 FSTAT5 FSTAT6 FSTAT7 FSTAT8
6N
74151
gnd G, H, I, J pin 9 thru 16
5K
02
/GDETSTAT
9B
SHT 7 (8-A)
I1 I3 I5 I7 J1 J3 J5 J7
FSTAT9 FSTAT10 FSTAT11 FSTAT12 FSTAT13 FSTAT14 FSTAT15 FSTAT16
6P
SHT8 (7-B)
RMAR6
5F
S04
6L
6M
DM 74154
SHT7 (3-A) 16
/GDETINIT
/GINIT1 /GINIT2 /GINIT3 /GINIT4 /GINIT5 /GINIT6 /GINIT7 /GINIT8 /GINIT9 /GINIT10 /GINIT11 /GINIT12 /GINIT13 /GINIT14 /GINIT15 /GINIT16
G2 G4 G6 G8 H2 H4 H6 H8 I2 I4 I6 I8 J2 J4 J6 J8
DF-8 REV. 7-75 V800

NEXT ASSY | USED ON

APPLICATION

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2° ±0

MATERIAL

FINISH—SEE NOTE

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

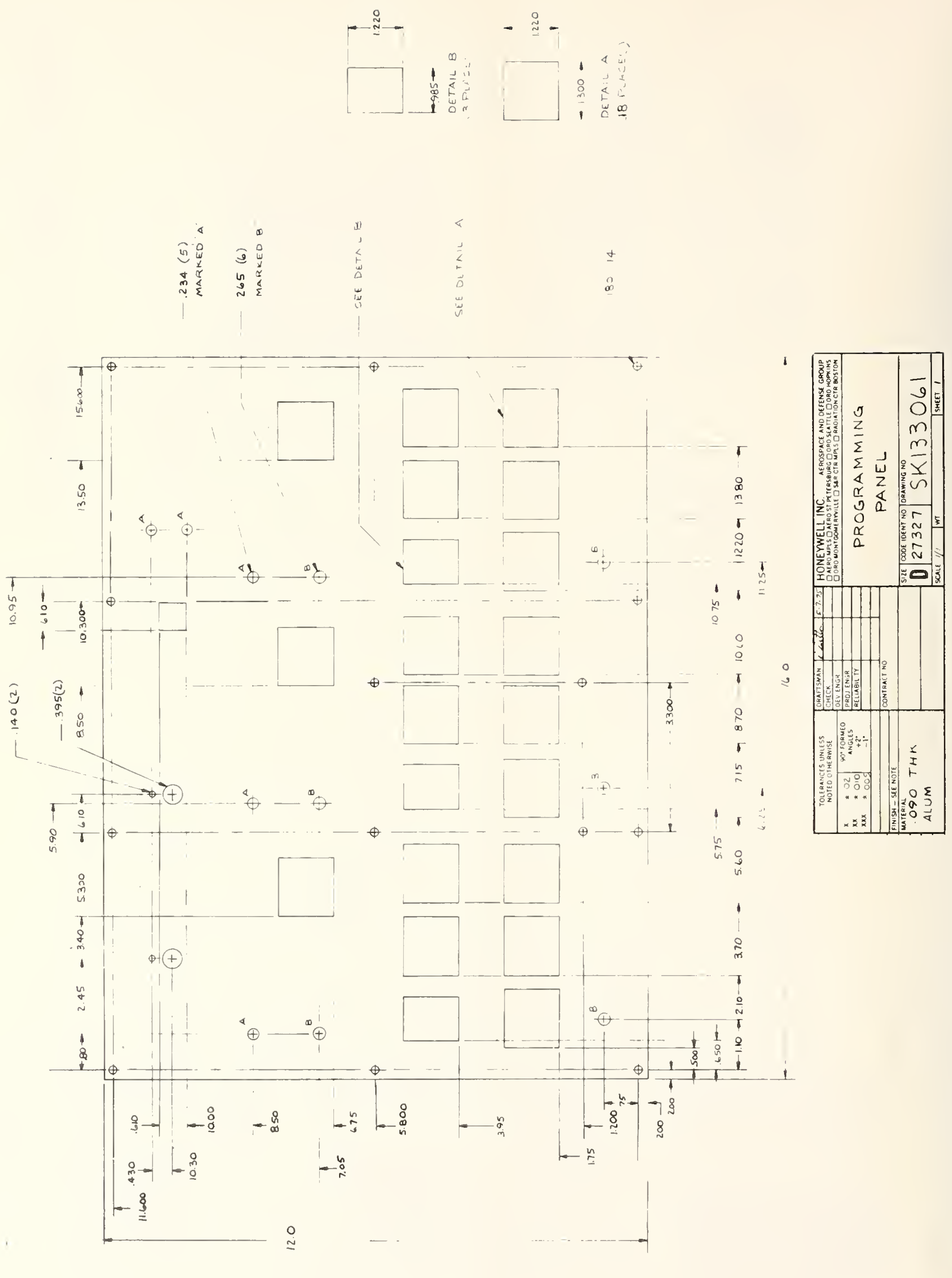
HONEYWELL INC.

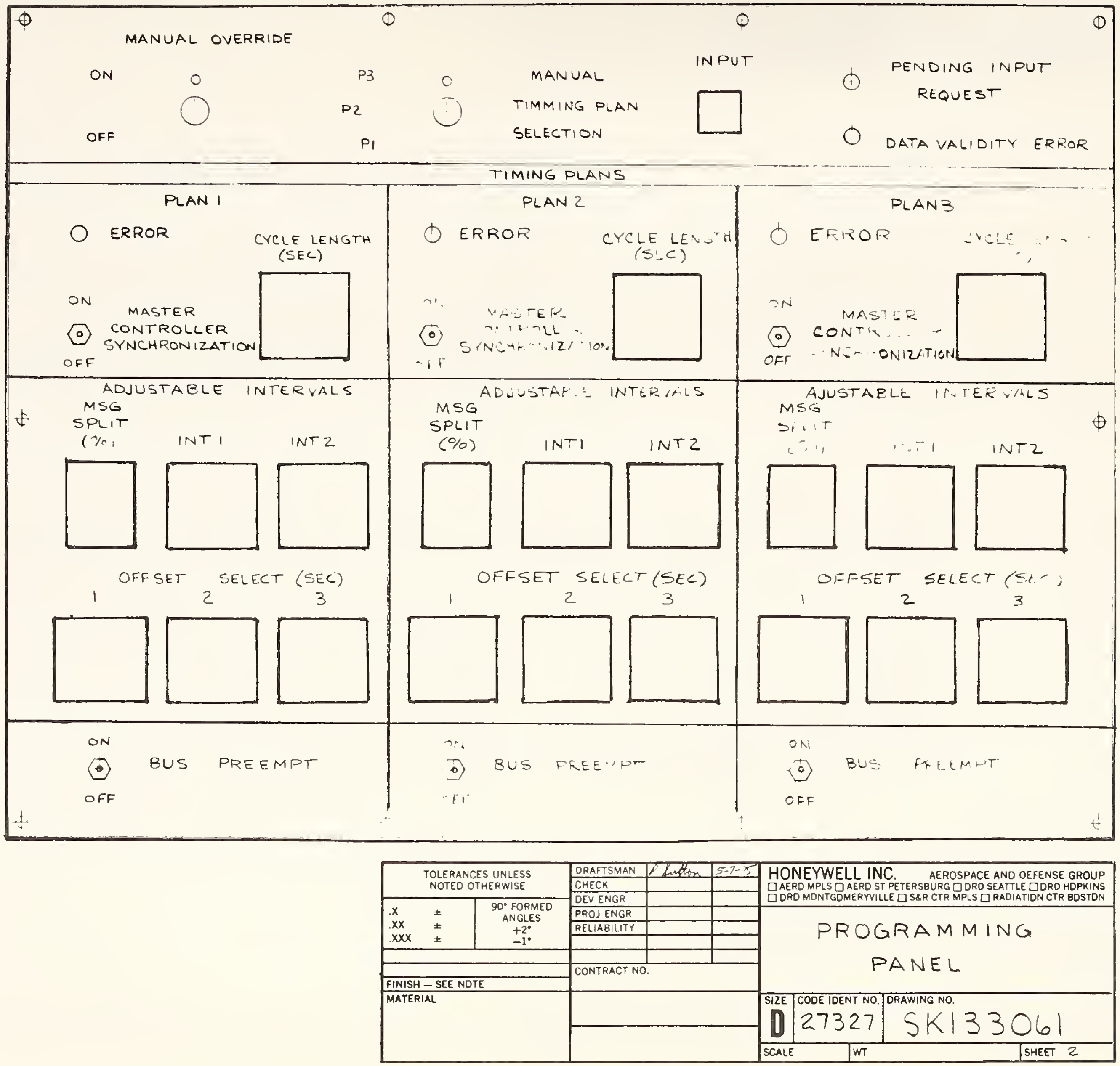
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

DETECTOR INTERFACE
BUS DETECTOR - CARD #2

SIZE C | CODE IDENT NO. 27327 | DRAWING NO.

SCALE | SHEET 9 OF 9

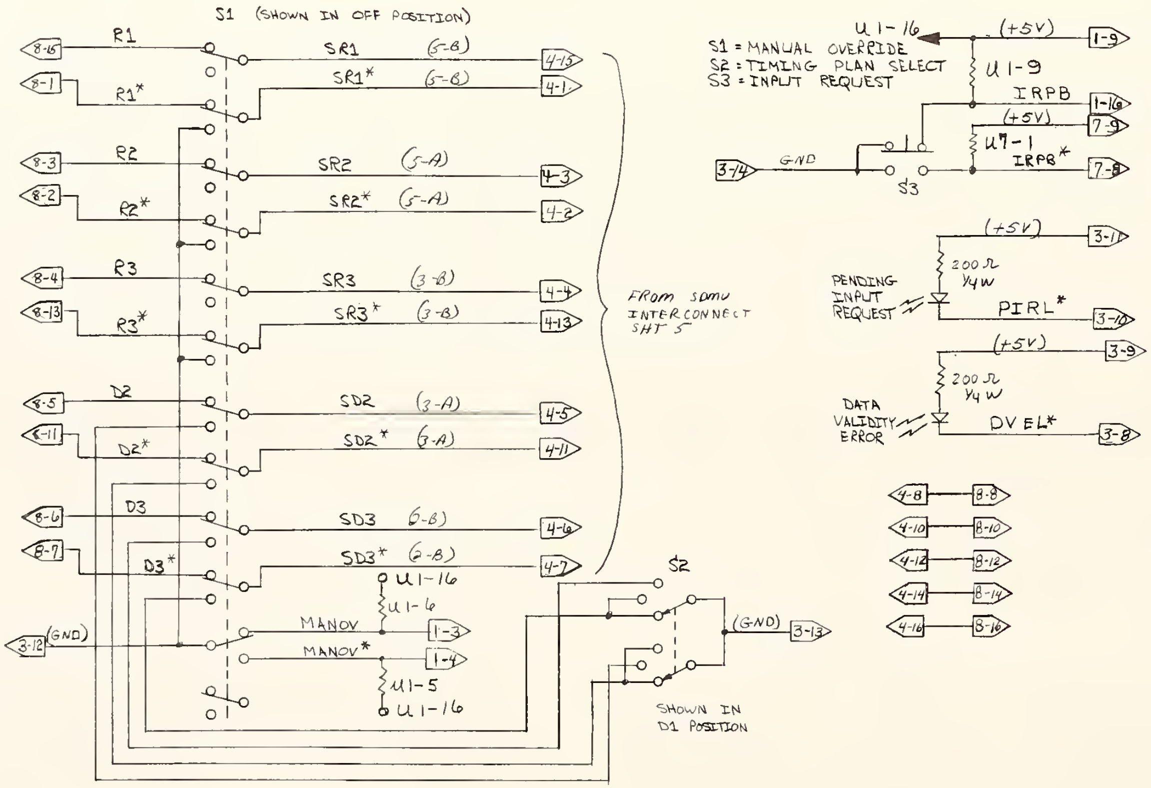

HONEYWELL INC. AEROSPACE AND DEFENSE GROUP
□ AERO MPLS □ AERO ST PETERSBURG □ ORO SEATTLE □ ORO HOPKINS
□ ORO MONTGOMERYVILLE □ S&R CTR MPLS □ RADIATION CTR BOSTON
PROGRAMMING PANEL
SIZE D
CODE IDENT NO 27327
DRAWING NO SK133061
SCALE 1/1
WT
SHEET 1
TOLERANCES UNLESS NOTED OTHERWISE
X ± .02
XX ± .010
XXX ± .005
90° FORMED ANGLES +2° -1°
FINISH – SEE NOTE
MATERIAL .090 THK ALUM
DRAFTSMAN
CHECK
DEV ENGR
PROJ ENGR
RELIABILITY
CONTRACT NO
5-7-75
.234 (5) MARKED A
.265 (6) MARKED B
SEE DETAIL B
SEE DETAIL A
.140 (2)
.395 (2)
DETAIL B 3 PLACES
DETAIL A 18 PLACES
1.220
.985
1.300
12.0
16.0
11.25
10.75
5.75
15.60
13.50
10.95
.610
10.300
8.50
5.90
5.300
3.40
2.45
.80
1.10
2.10
3.70
5.60
7.15
8.70
10.10
12.20
13.80
3.300
.500
.650
11.600
.430
.610
10.30
10.00
8.50
7.05
6.75
5.800
3.95
1.75
1.200
.75
.200
.200
A
B


MANUAL OVERRIDE
ON
OFF
P3
P2
P1
MANUAL
TIMMING PLAN
SELECTION
INPUT
PENDING INPUT
REQUEST
DATA VALIDITY ERROR
TIMING PLANS
PLAN 1
ERROR
CYCLE LENGTH (SEC)
ON
MASTER CONTROLLER SYNCHRONIZATION
OFF
ADJUSTABLE INTERVALS
MSG SPLIT (%)
INT 1
INT 2
OFFSET SELECT (SEC)
1
2
3
BUS PREEMPT
PLAN 2
PLAN 3
TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES +2° −1°
FINISH — SEE NOTE
MATERIAL
DRAFTSMAN
5-7-75
CHECK
DEV ENGR
PROJ ENGR
RELIABILITY
CONTRACT NO.
HONEYWELL INC. AEROSPACE AND DEFENSE GROUP
□ AERO MPLS □ AERO ST PETERSBURG □ ORD SEATTLE □ ORD HOPKINS
□ ORD MONTGOMERYVILLE □ S&R CTR MPLS □ RADIATION CTR BOSTON
PROGRAMMING PANEL
SIZE D
CODE IDENT NO. 27327
DRAWING NO. SK133061
SCALE
WT
SHEET 2

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



| | | | | |
|---|---|---|---|---|
| TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° ±1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 | |
| MATERIAL | CONTRACT NO. | PROGRAM PANEL | | |
| NEXT ASSY | USED ON | SIZE C | CODE IDENT NO. 27327 | DRAWING NO. |
| APPLICATION | FINISH–SEE NOTE | SCALE | | SHEET 1 OF 13 |

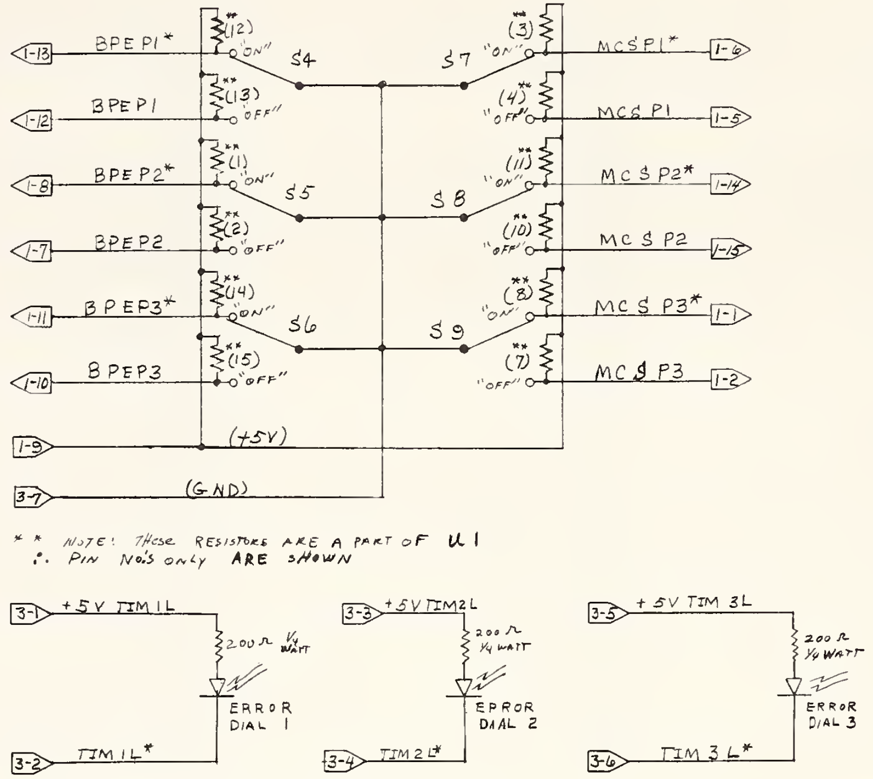
DF-5 REV. 7-75 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
|  |  |  |  |  |



** NOTE: These RESISTORS ARE A PART OF U1
∴ PIN NO.S ONLY ARE SHOWN

S 4 = BUS PREEMPT DIAL 1

S5 = BUS PREEMPT DAIL 2

S6 = BUS PREEMPT DIAL 3

S 7 = MASTER CONTROLLER SNYC DAIL 1

S8 = MASTER CONTROLLER SNYC. DIAL 2

S 9 = MASTER CONTROLLER SNYC. DIAL 3

NOTE: ALL INPUTS AND OUTPUTS COME FROM OR GO TO the PANEL ELECTRONICS CARD

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2° ±0

MATERIAL

FINISH—SEE NOTE

NEXT ASSY | USED ON

APPLICATION

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PROGRAM PANEL

SIZE C | CODE IDENT NO. 27327 | DRAWING NO.

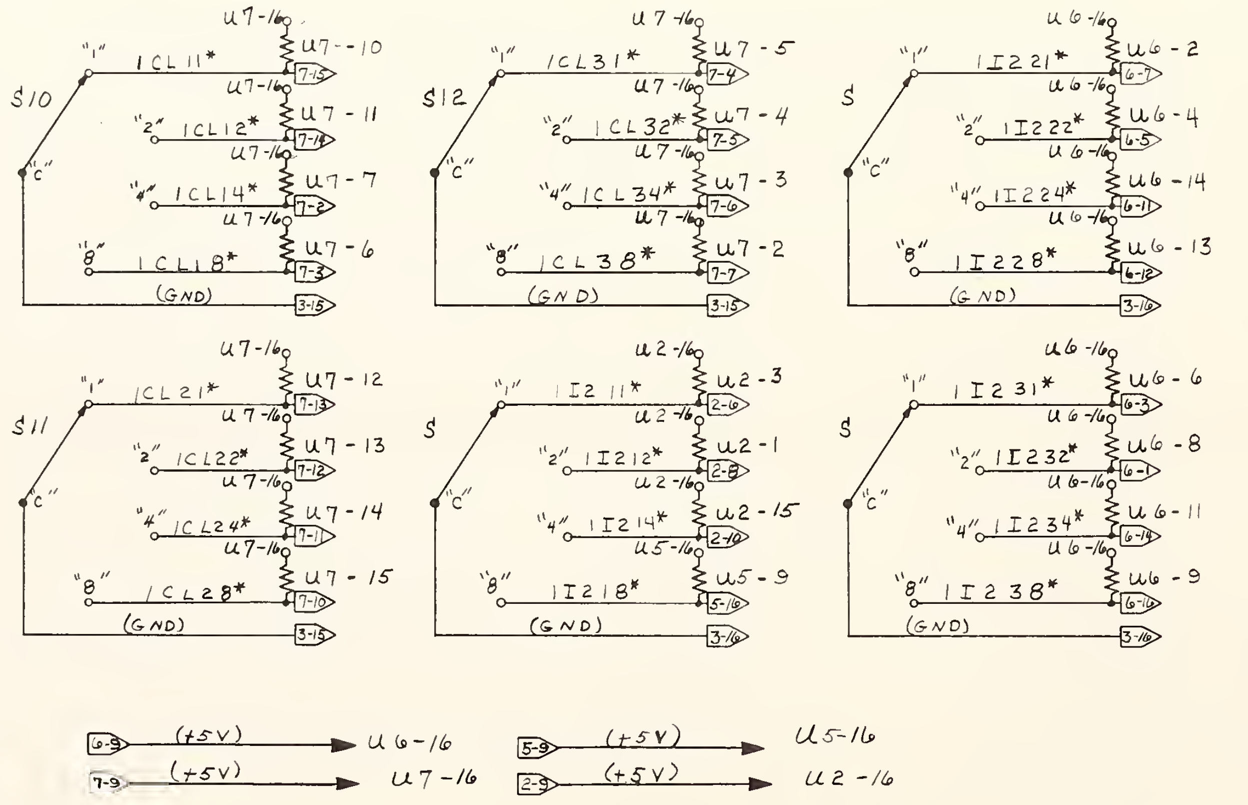
SCALE | SHEET 2 OF 13

DP-8 REV. 7-70 V000

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



S10
"1" 1CL11* U7-16 U7--10 7-15
"2" 1CL12* U7-16 U7-11 7-14
"4" 1CL14* U7-16 U7-7 7-2
"8" 1CL18* U7-16 U7-6 7-3
"C"
(GND) 3-15
S12
"1" 1CL31* U7-16 U7-5 7-4
"2" 1CL32* U7-16 U7-4 7-5
"4" 1CL34* U7-16 U7-3 7-6
"8" 1CL38* U7-16 U7-2 7-7
"C"
(GND) 3-15
S
"1" 1I221* U6-16 U6-2 6-7
"2" 1I222* U6-16 U6-4 6-5
"4" 1I224* U6-16 U6-14 6-11
"8" 1I228* U6-16 U6-13 6-12
"C"
(GND) 3-16
S11
"1" 1CL21* U7-16 U7-12 7-13
"2" 1CL22* U7-16 U7-13 7-12
"4" 1CL24* U7-16 U7-14 7-1
"8" 1CL28* U7-16 U7-15 7-10
"C"
(GND) 3-15
S
"1" 1I211* U2-16 U2-3 2-6
"2" 1I212* U2-16 U2-1 2-8
"4" 1I214* U2-16 U2-15 2-10
"8" 1I218* U5-16 U5-9 5-16
"C"
(GND) 3-16
S
"1" 1I231* U6-16 U6-6 6-3
"2" 1I232* U6-16 U6-8 6-1
"4" 1I234* U6-16 U6-11 6-14
"8" 1I238* U6-16 U6-9 6-16
"C"
(GND) 3-16
6-9 (+5V) U6-16
7-9 (+5V) U7-16
5-9 (+5V) U5-16
2-9 (+5V) U2-16

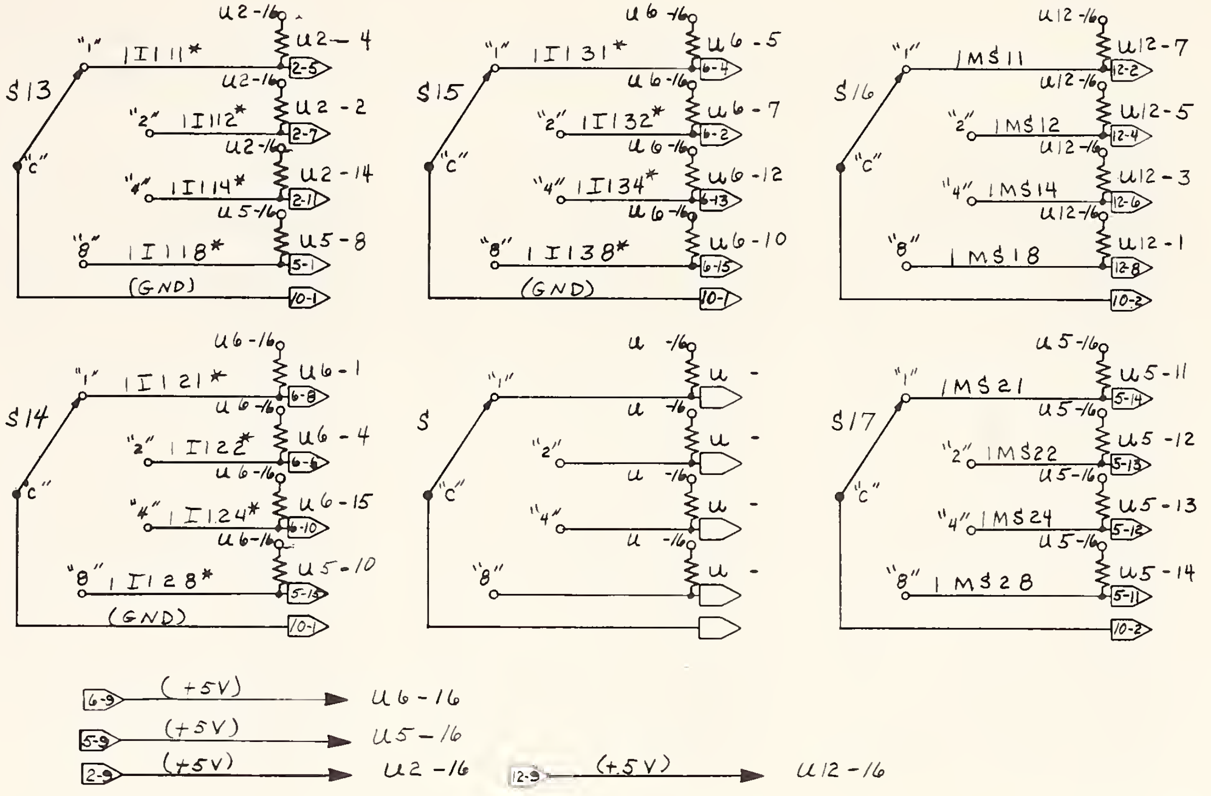
| | | TOLERANCES UNLESS NOTED OTHERWISE | DRAFTSMAN | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|
| | | .X ± .XX ± .XXX ± 90° FORMED ANGLES ±2° ±1° | CHECKER | PROGRAM PANEL | |
| | | | DEV ENGR | | |
| | | | ENGRG MGT | | |
| | | MATERIAL | CONTRACT NO. | | |
| | | | | SIZE C | CODE IDENT NO. 27327 | DRAWING NO. |
| NEXT ASSY | USED ON | | | | |
| APPLICATION | | FINISH—SEE NOTE | | SCALE | SHEET 3 OF 13 |

DF-8 REV. 7-75 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |



S13
"1" 1I111* u2-16 u2-4 2-5
"2" 1I112* u2-16 u2-2 2-7
"4" 1I114* u2-16 u2-14 2-1
"8" 1I118* u5-16 u5-8 5-1
"C"
(GND) 10-1
S15
"1" 1I131* u6-16 u6-5 6-4
"2" 1I132* u6-16 u6-7 6-2
"4" 1I134* u6-16 u6-12 6-13
"8" 1I138* u6-16 u6-10 6-15
"C"
(GND) 10-1
S16
"1" 1MS11 u12-16 u12-7 12-2
"2" 1MS12 u12-16 u12-5 12-4
"4" 1MS14 u12-16 u12-3 12-6
"8" 1MS18 u12-16 u12-1 12-8
"C"
10-2
S14
"1" 1I121* u6-16 u6-1 6-8
"2" 1I122* u6-16 u6-4 6-6
"4" 1I124* u6-16 u6-15 6-10
"8" 1I128* u6-16 u5-10 5-15
"C"
(GND) 10-1
S
"1" u -16 u -
"2" u -16 u -
"4" u -16 u -
"8" u -16 u -
"C"
S17
"1" 1MS21 u5-16 u5-11 5-14
"2" 1MS22 u5-16 u5-12 5-13
"4" 1MS24 u5-16 u5-13 5-12
"8" 1MS28 u5-16 u5-14 5-11
"C"
10-2
6-9 (+5V) u6-16
5-9 (+5V) u5-16
2-9 (+5V) u2-16
12-9 (+5V) u12-16

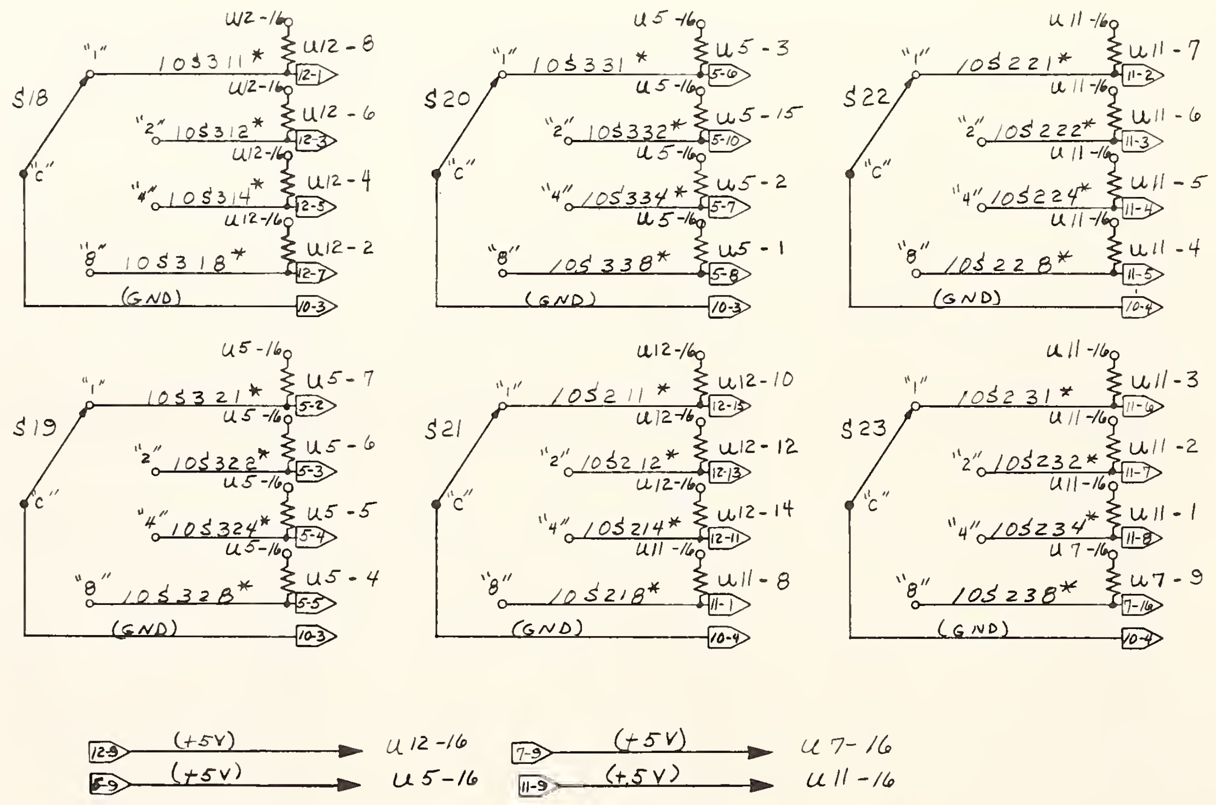
| | | | | |
|---|---|---|---|---|
| | | TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES +2° -1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413 |
| | | MATERIAL | CONTRACT NO. | PROGRAM PANEL |
| NEXT ASSY | USED ON | | | SIZE C CODE IDENT NO. 27327 DRAWING NO. |
| APPLICATION | | FINISH—SEE NOTE | | SCALE SHEET 4 OF 13 |

DF-5 REV. 7-75 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



S18
"1" 10S311* U12-16 U12-8 12-1
"2" 10S312* U12-16 U12-6 12-3
"4" 10S314* U12-16 U12-4 12-5
"8" 10S318* U12-16 U12-2 12-7
"C"
(GND) 10-3
S20
"1" 10S331* U5-16 U5-3 5-6
"2" 10S332* U5-16 U5-15 5-10
"4" 10S334* U5-16 U5-2 5-7
"8" 10S338* U5-16 U5-1 5-8
"C"
(GND) 10-3
S22
"1" 10S221* U11-16 U11-7 11-2
"2" 10S222* U11-16 U11-6 11-3
"4" 10S224* U11-16 U11-5 11-4
"8" 10S228* U11-16 U11-4 11-5
"C"
(GND) 10-4
S19
"1" 10S321* U5-16 U5-7 5-2
"2" 10S322* U5-16 U5-6 5-3
"4" 10S324* U5-16 U5-5 5-4
"8" 10S328* U5-16 U5-4 5-5
"C"
(GND) 10-3
S21
"1" 10S211* U12-16 U12-10 12-15
"2" 10S212* U12-16 U12-12 12-13
"4" 10S214* U12-16 U12-14 12-11
"8" 10S218* U11-16 U11-8 11-1
"C"
(GND) 10-4
S23
"1" 10S231* U11-16 U11-3 11-6
"2" 10S232* U11-16 U11-2 11-7
"4" 10S234* U7-16 U11-1 11-8
"8" 10S238* U7-16 U7-9 7-16
"C"
(GND) 10-4
12-9 (+5V) U12-16
5-9 (+5V) U5-16
7-9 (+5V) U7-16
11-9 (+5V) U11-16

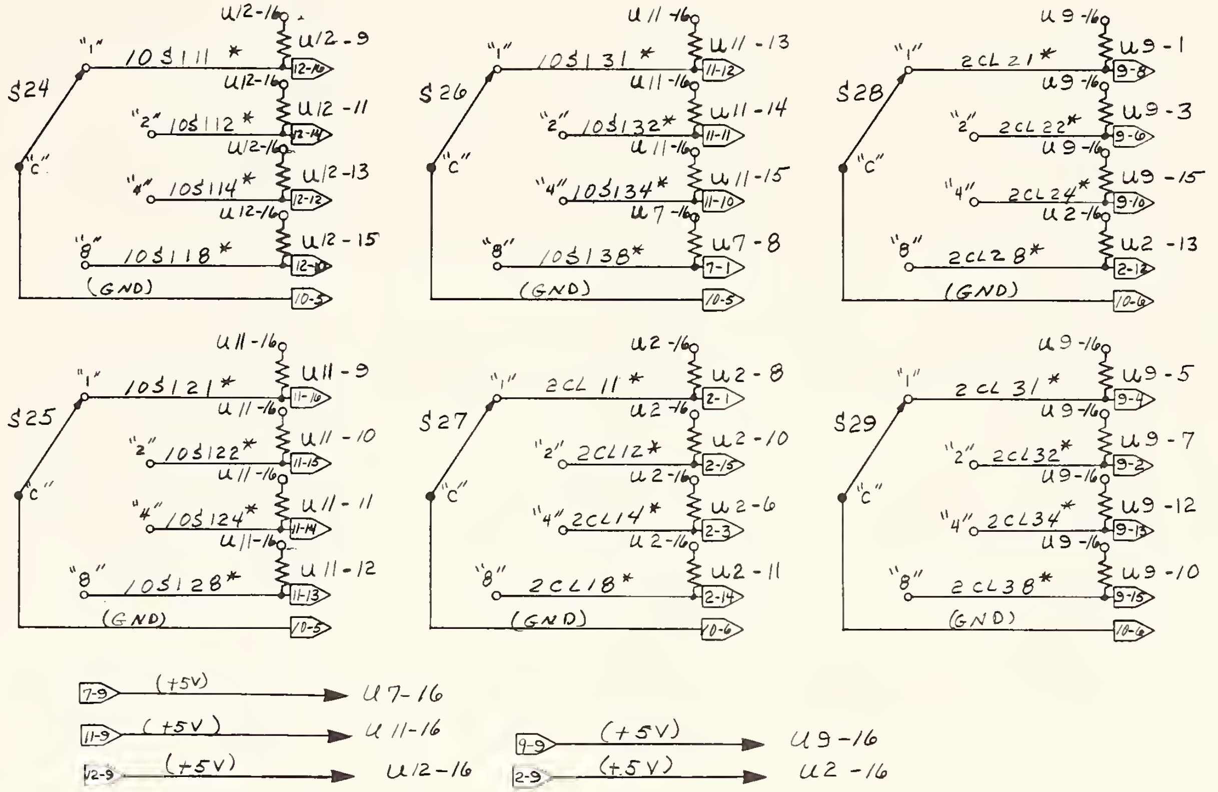
| | | TOLERANCES UNLESS NOTED OTHERWISE | DRAFTSMAN | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|
| | | .X ± .XX ± .XXX ± | CHECKER | PROGRAM PANEL | |
| | | 90° FORMED ANGLES ±2° ±1° | DEV ENGR | | |
| | | | ENGRG MGT | | |
| | | MATERIAL | CONTRACT NO | | |
| NEXT ASSY | USED ON | | | SIZE C | CODE IDENT NO. 27327 | 
| APPLICATION | | FINISH–SEE NOTE | | SCALE | DRAWING NO. SHEET 5 OF 13 |

DF-8 REV. 7-78 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



| | | TOLERANCES UNLESS NOTED OTHERWISE | DRAFTSMAN | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|
| | | .X ±<br>.XX ±<br>.XXX ± | 90° FORMED ANGLES ±2° ±10 | CHECKER<br>DEV ENGR<br>ENGRG MGT | PROGRAM PANEL |
| | | MATERIAL | CONTRACT NO. | SIZE C | CODE IDENT NO. 27327 | DRAWING NO. |
| NEXT ASSY | USED ON | | | | |
| APPLICATION | | FINISH–SEE NOTE | | SCALE | SHEET 6 OF 13 |

DF-8 REV. 7-76 V800

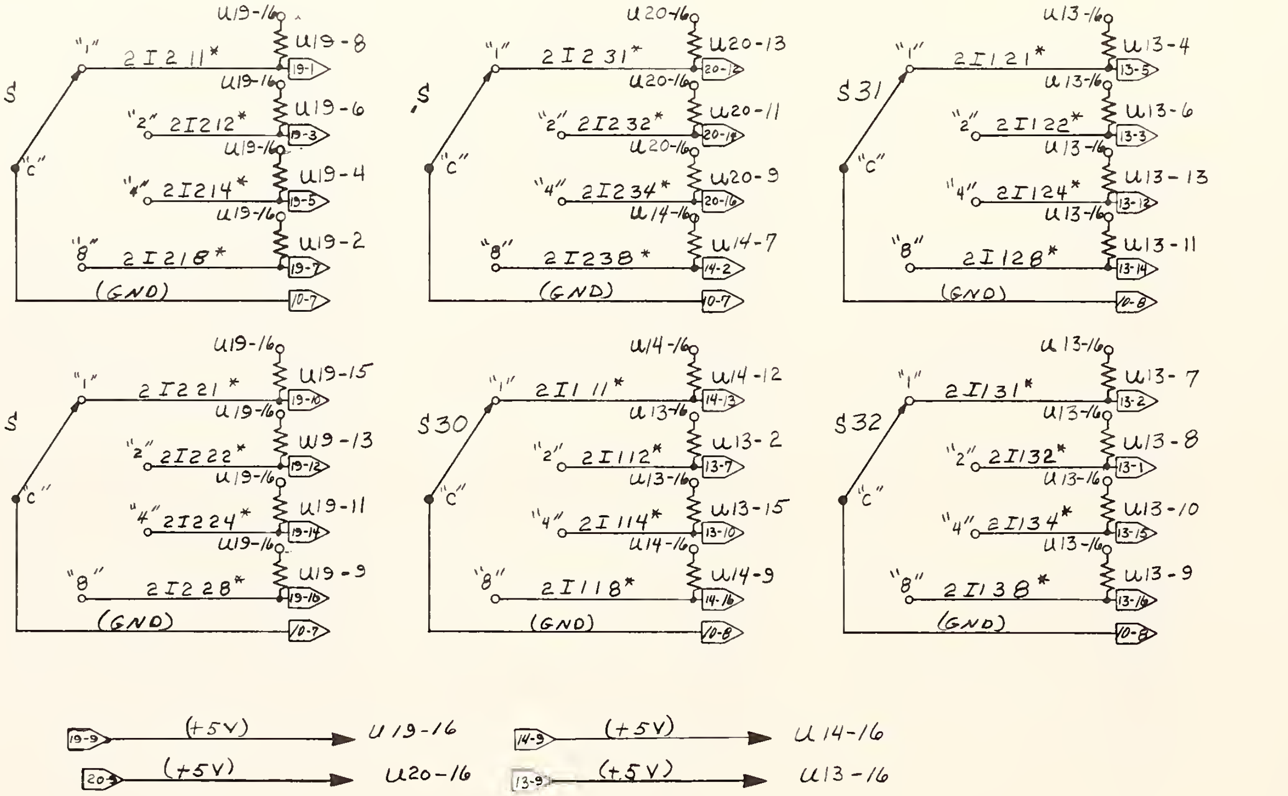
5 4 3 2 1

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |



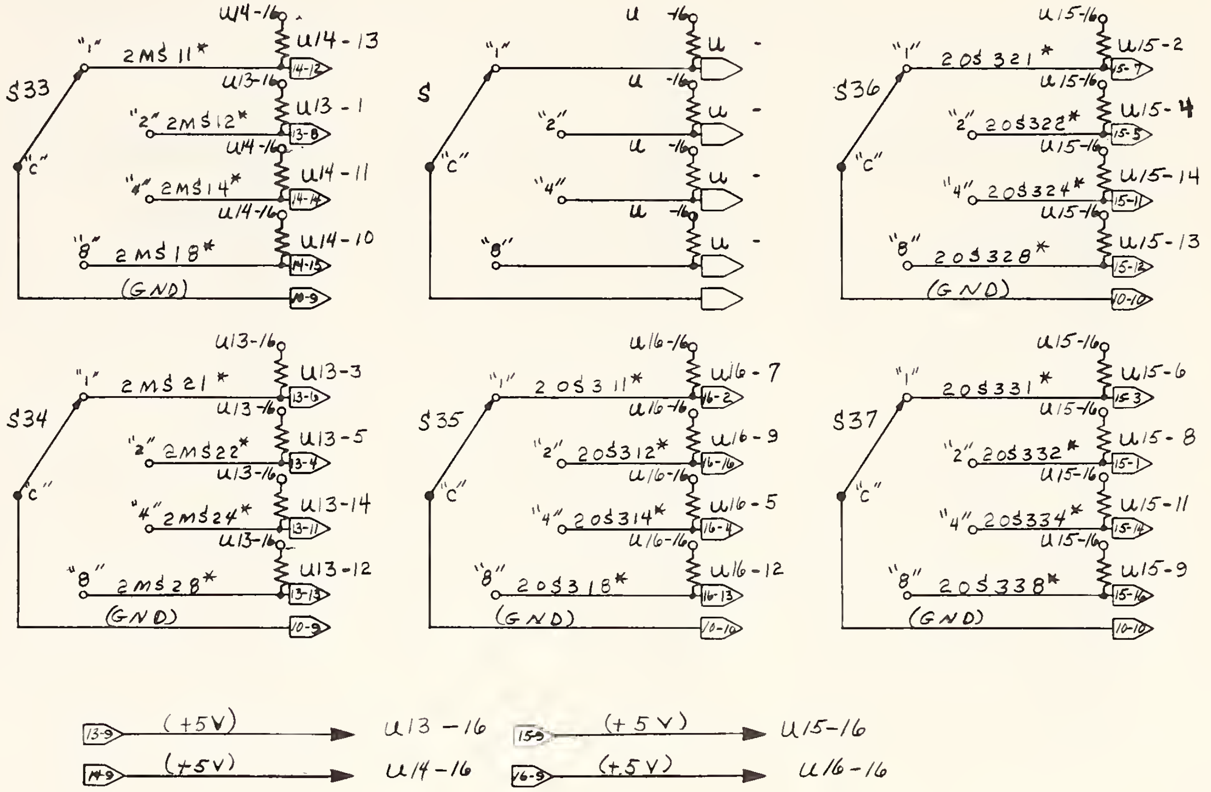
| | | TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2°<br>±1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|
| | | MATERIAL | CONTRACT NO. | PROGRAM PANEL | |
| NEXT ASSY | USED ON | | | SIZE C | CODE IDENT NO. 27327 | DRAWING NO. |
| APPLICATION | | FINISH–SEE NOTE | | SCALE | SHEET 7 OF 13 |

DF-6 REV. 7-78 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |



| | | | |
|---|---|---|---|
| TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° -1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
| MATERIAL | CONTRACT NO. | PROGRAM PANEL | |
| NEXT ASSY / USED ON<br>APPLICATION | FINISH—SEE NOTE | SIZE C | CODE IDENT NO. 27327 / DRAWING NO. |
| | | SCALE | SHEET 8 OF 13 |

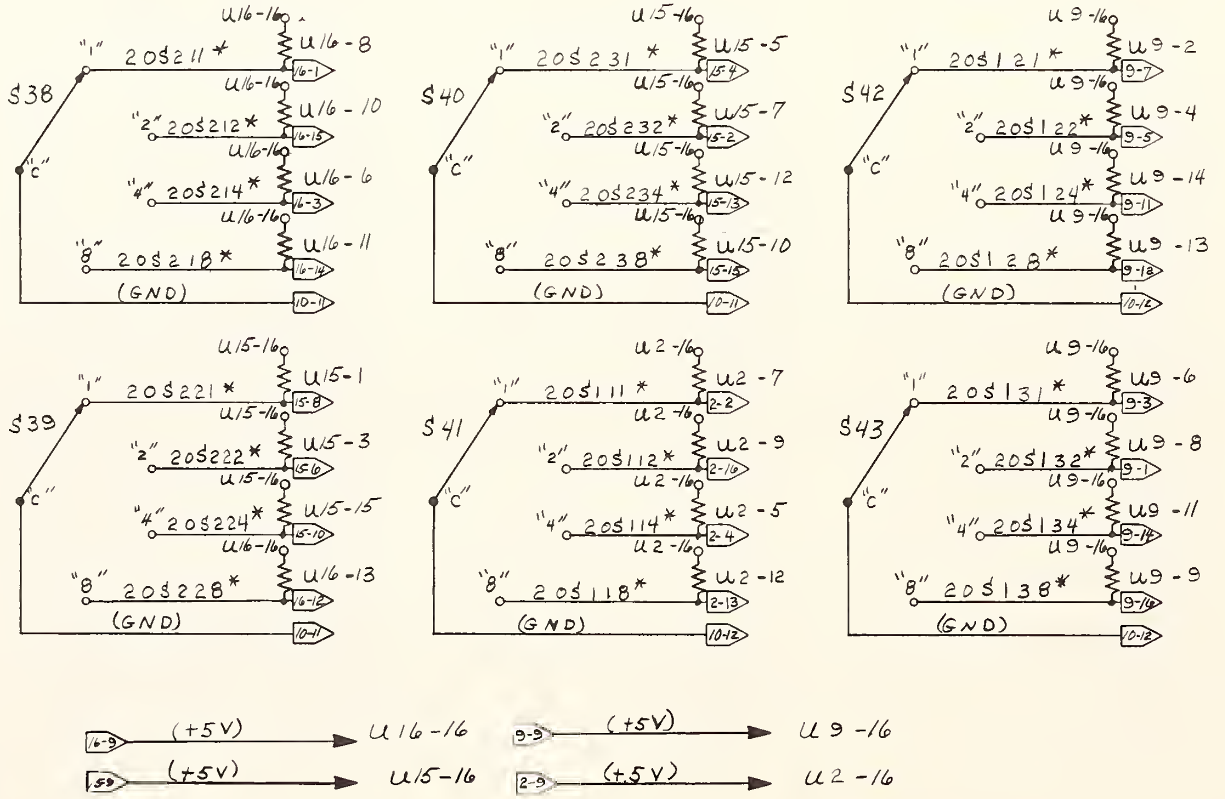
DF-8 REV. 7-75 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |



| | |
|---|---|
| NEXT ASSY | USED ON |
| APPLICATION | |

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2° -1°
MATERIAL
FINISH–SEE NOTE

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PROGRAM PANEL

SIZE C
CODE IDENT NO. 27327
DRAWING NO.
SCALE
SHEET 9 OF 13

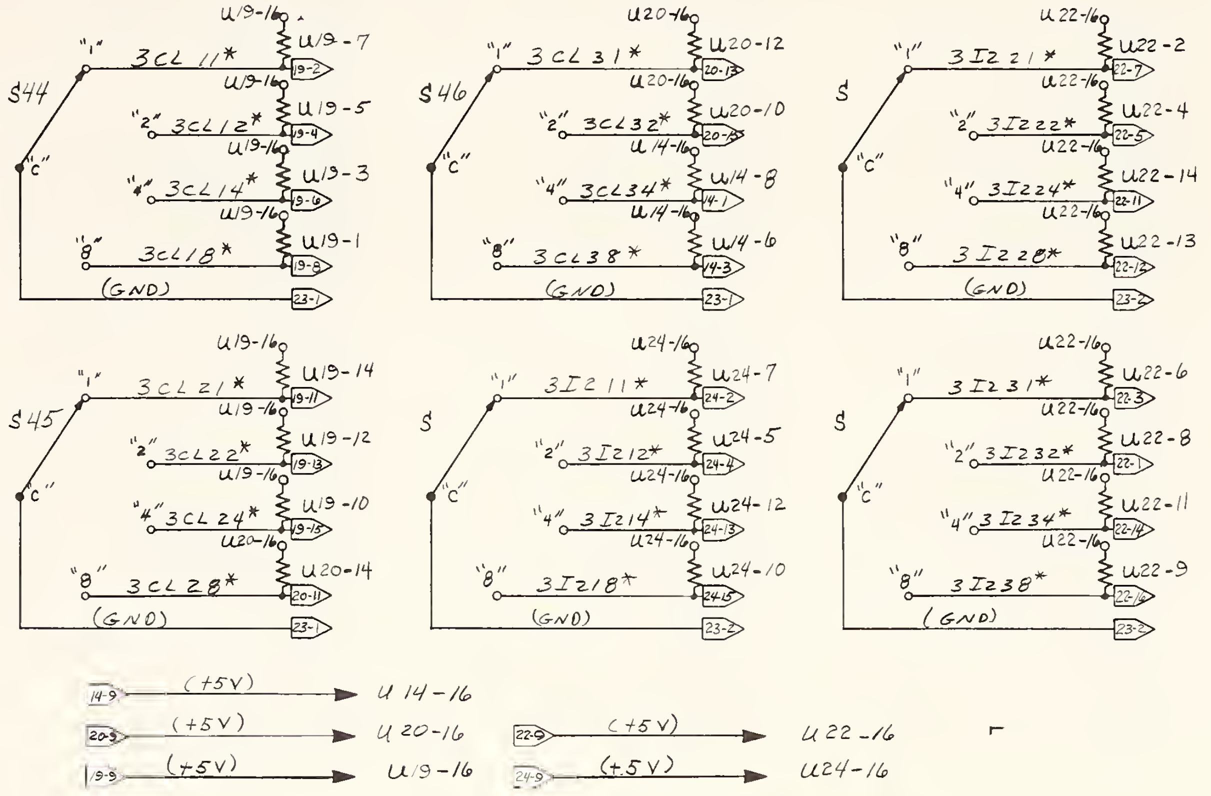
DF-8 REV. 7-78 V600

5 4 3 2 1

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



C B A

| | | TOLERANCES UNLESS NOTED OTHERWISE<br>X ±<br>XX ±<br>XXX ±<br>90° FORMED ANGLES ±2° ±1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|
| | | MATERIAL | CONTRACT NO. | PROGRAM PANEL | |
| NEXT ASSY | USED ON | | | SIZE C | CODE IDENT NO. 27327 |
| APPLICATION | | FINISH–SEE NOTE | | SCALE | SHEET 10 OF 13 |

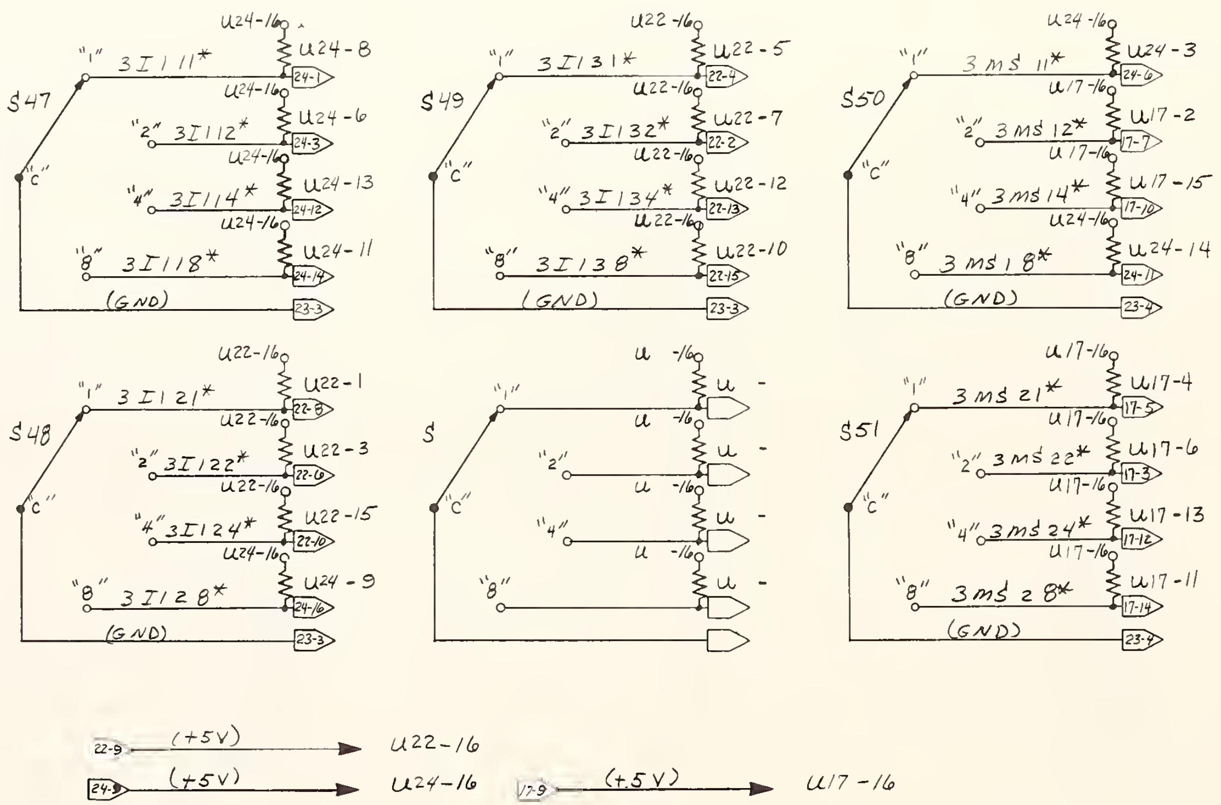
DRAWING NO.

DF-5 REV. 7-75 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |



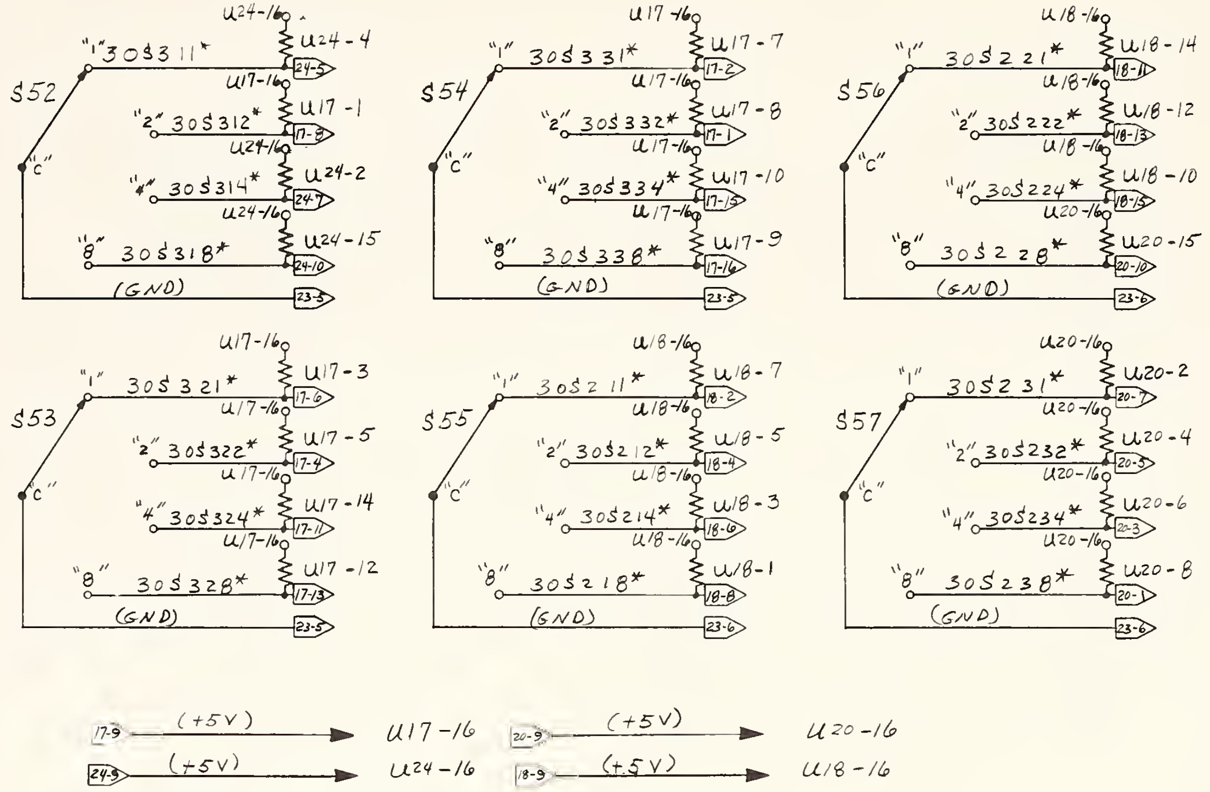
| | | | | |
|---|---|---|---|---|
| TOLERANCES UNLESS NOTED OTHERWISE<br>X ±<br>XX ±<br>XXX ±<br>90° FORMED ANGLES ±2° ±1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 | |
| MATERIAL | CONTRACT NO. | PROGRAM PANEL | | |
| NEXT ASSY | USED ON | SIZE C | CODE IDENT NO. 27327 | DRAWING NO. |
| APPLICATION | FINISH—SEE NOTE | SCALE | | SHEET 11 OF 13 |

DF-8 REV. 7-78 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



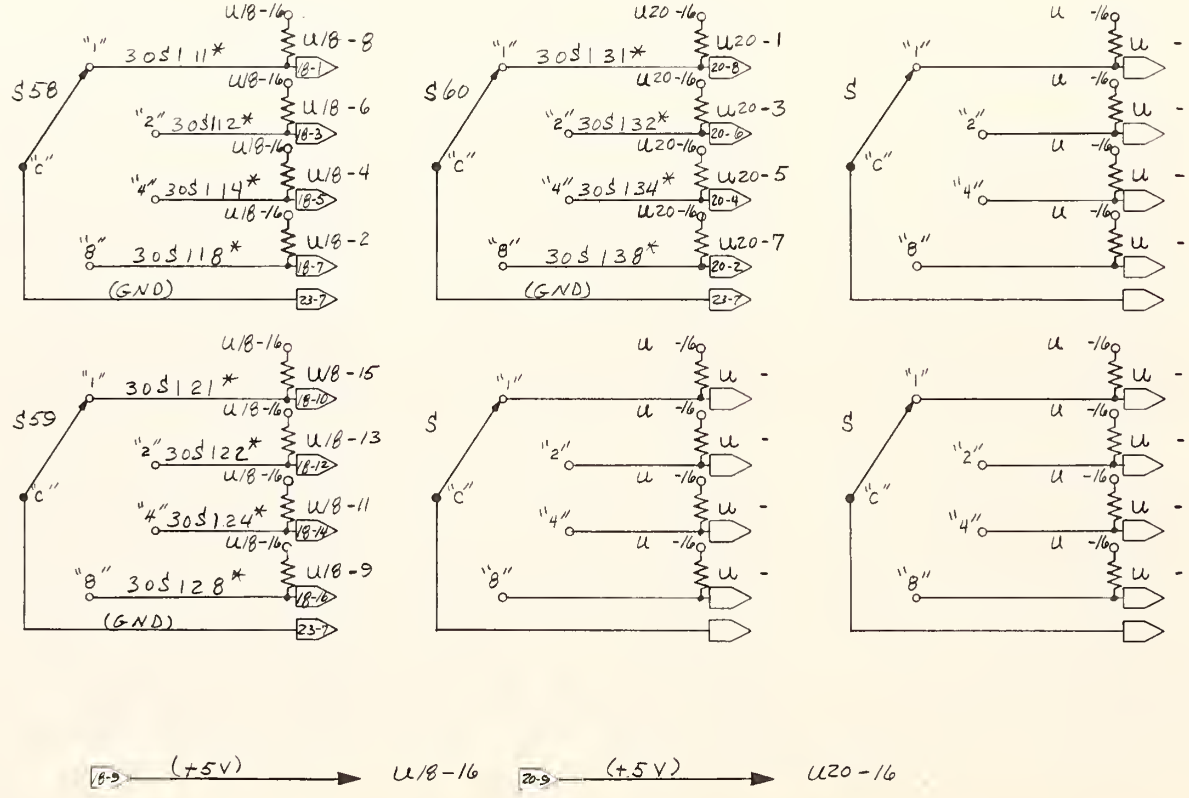
| | | TOLERANCES UNLESS NOTED OTHERWISE | | DRAFTSMAN | | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|---|---|
| | | X ±<br>XX ±<br>XXX ± | 90° FORMED ANGLES ±2°<br>±1° | CHECKER | | PROGRAM PANEL | |
| | | | | DEV ENGR | | | |
| | | | | ENGRG MGT | | | |
| | | MATERIAL | | CONTRACT NO. | | SIZE<br>C | CODE IDENT NO.<br>27327 |
| NEXT ASSY | USED ON | | | | | DRAWING NO. | |
| APPLICATION | | FINISH–SEE NOTE | | | | SCALE | SHEET 12 OF 13 |

DF-8 REV. 7-75 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



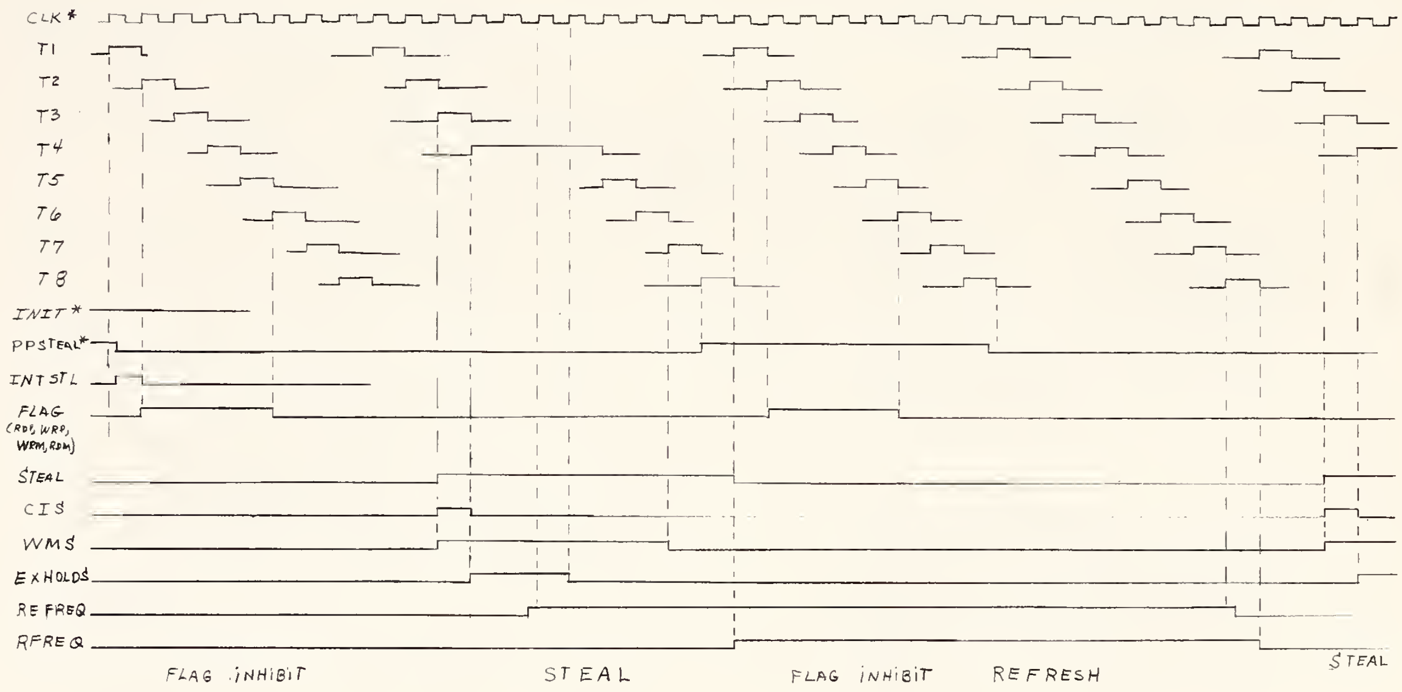
| | | TOLERANCES UNLESS NOTED OTHERWISE<br>X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° ±1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
|---|---|---|---|---|---|
| | | MATERIAL | CONTRACT NO. | PROGRAM PANEL | |
| NEXT ASSY | USED ON | | | SIZE C | CODE IDENT NO. 27327 | DRAWING NO. |
| APPLICATION | | FINISH–SEE NOTE | | SCALE | SHEET 13 OF 13 |

DF-8 REV. 7-76 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



| | | | |
|---|---|---|---|
| NEXT ASSY | USED ON | TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° -1°<br>MATERIAL<br>FINISH—SEE NOTE | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT<br>CONTRACT NO. |
| APPLICATION | | | |

HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

"STEAL" CIRCUIT FOR IMP 16 TIMING

SIZE C | CODE IDENT NO. 27327 | DRAWING NO.

SCALE | SHEET 1 OF 1

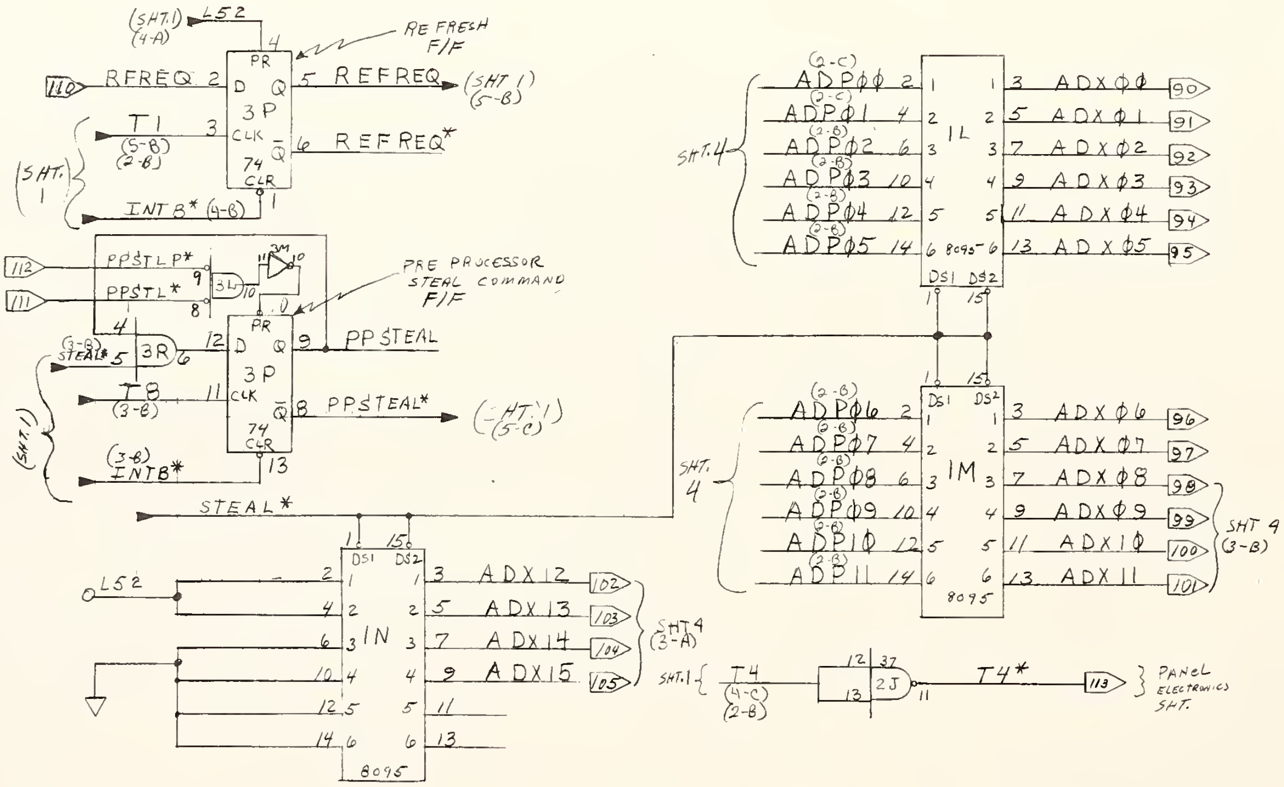
DF-8 REV. 7-75 V800

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

REVISIONS

| ZONE | LTR | OESCRIPTION | OATE | APPROVED |
|---|---|---|---|---|
| | | | | |



| | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| NEXT ASSY | USED ON | | | |
| APPLICATION | | | | |

| TOLERANCES UNLESS NOTED OTHERWISE | | DRAFTSMAN | |
|---|---|---|---|
| .X ± <br> .XX ± <br> .XXX ± | 90° FORMED ANGLES ±2° ±1° | CHECKER | |
| | | DEV ENGR | |
| | | ENGRG MGT | |
| MATERIAL | | CONTRACT NO. | |
| FINISH–SEE NOTE | | | |

**HONEYWELL INC.**

SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

"CYCLE STEAL" FOR
IMP 16 (CONTROL/ADDRESS)

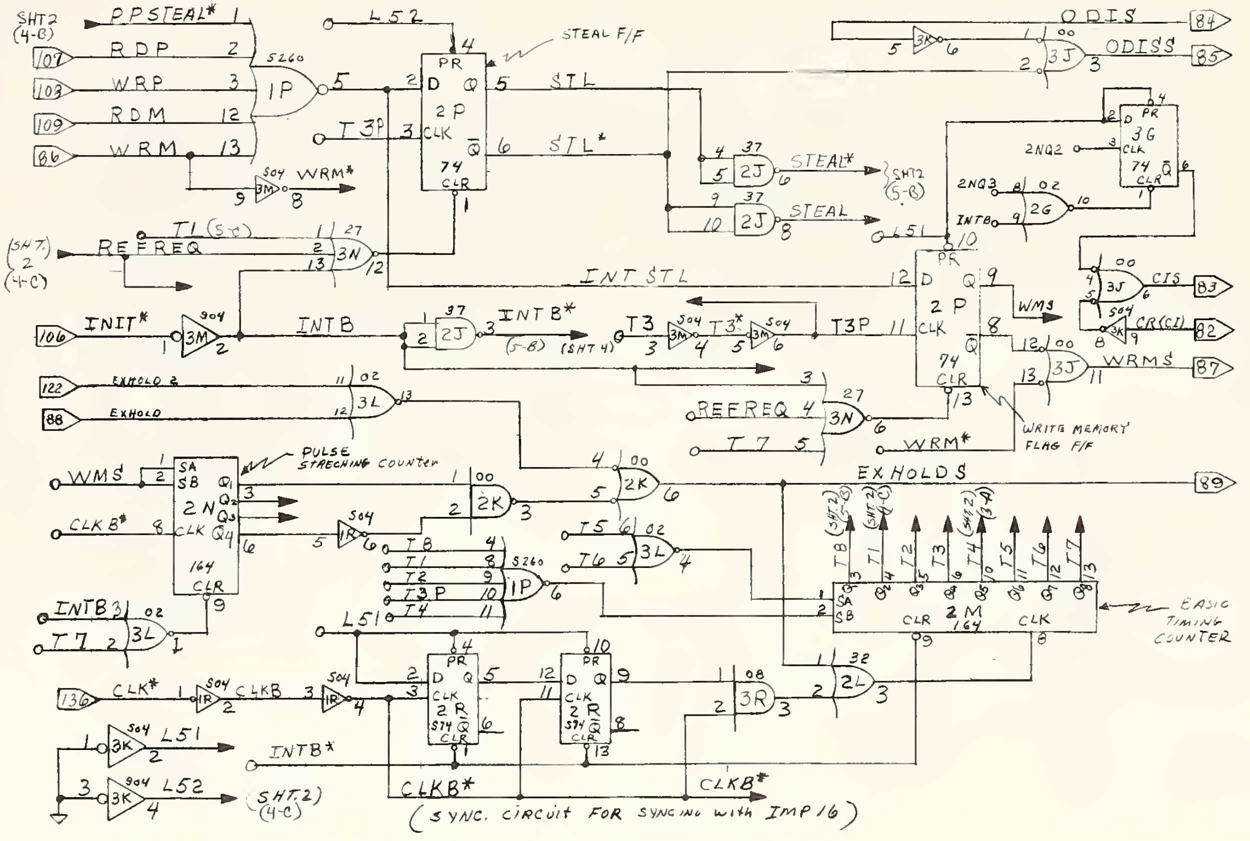
| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| C | 27327 | |
| SCALE | | SHEET 2 OF 4 |

DF-5 REV. 7-78 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |



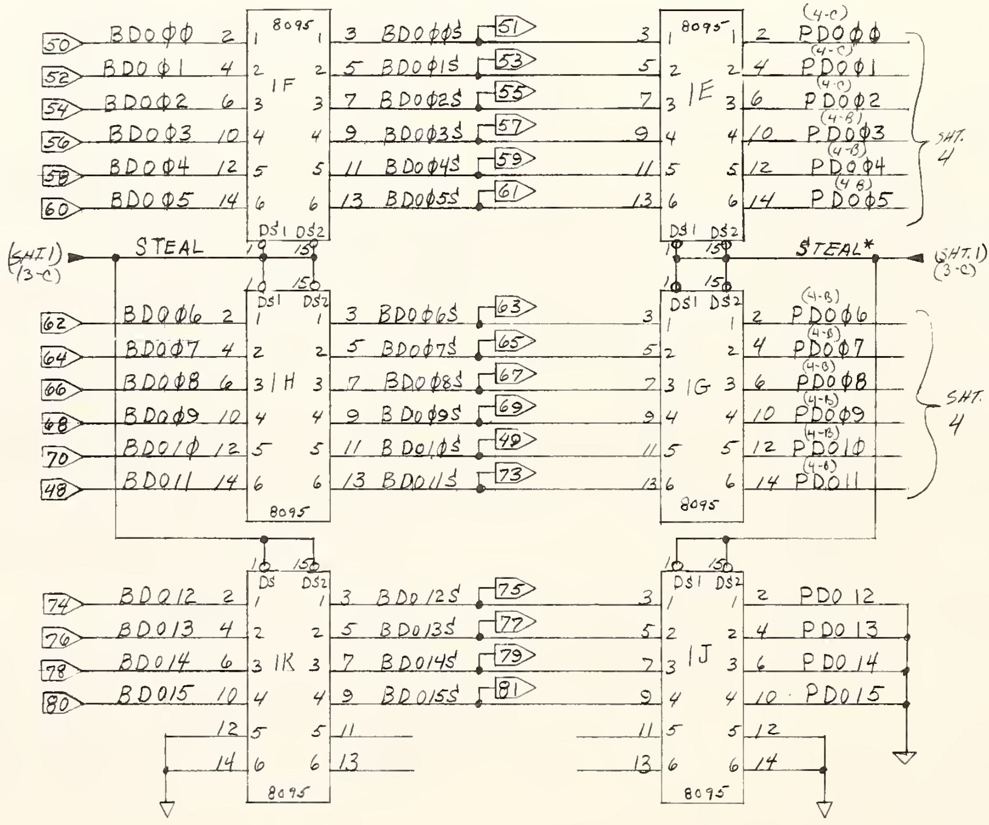
| | | |
|---|---|---|
| TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° ±0 | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413 |
| MATERIAL | CONTRACT NO. | "CYCLE STEAL" FOR IMP 16 (CONTROL) |
| NEXT ASSY / USED ON / APPLICATION | FINISH–SEE NOTE | SIZE C CODE IDENT NO. 27327 DRAWING NO. |
| | | SCALE SHEET 1 OF 4 |

OF-8 REV. 7-78 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



| PART NO. | LOCATION |
|---|---|
| 7400 | 3J |
| 74S00 | 2K |
| 7402 | 3L |
| 74S04 | 1R, 3K, 3M |
| 7408 | 3R |
| 7427 | 3N |
| 7432 | 2L |
| 7437 | 2J |
| 7474 | 2P, 3P, 3G |
| 74S74 | 2R |
| 74S260 | 1P |
| DM8570 | 2M, 2N |
| DM 8095 | 1E, 1F, 1G, 1H,<br>1J, 1K, 1L, 1M, 1N |
| 74174 | 2E, 2F, 2H, 3H |
| 5623 | 1D |

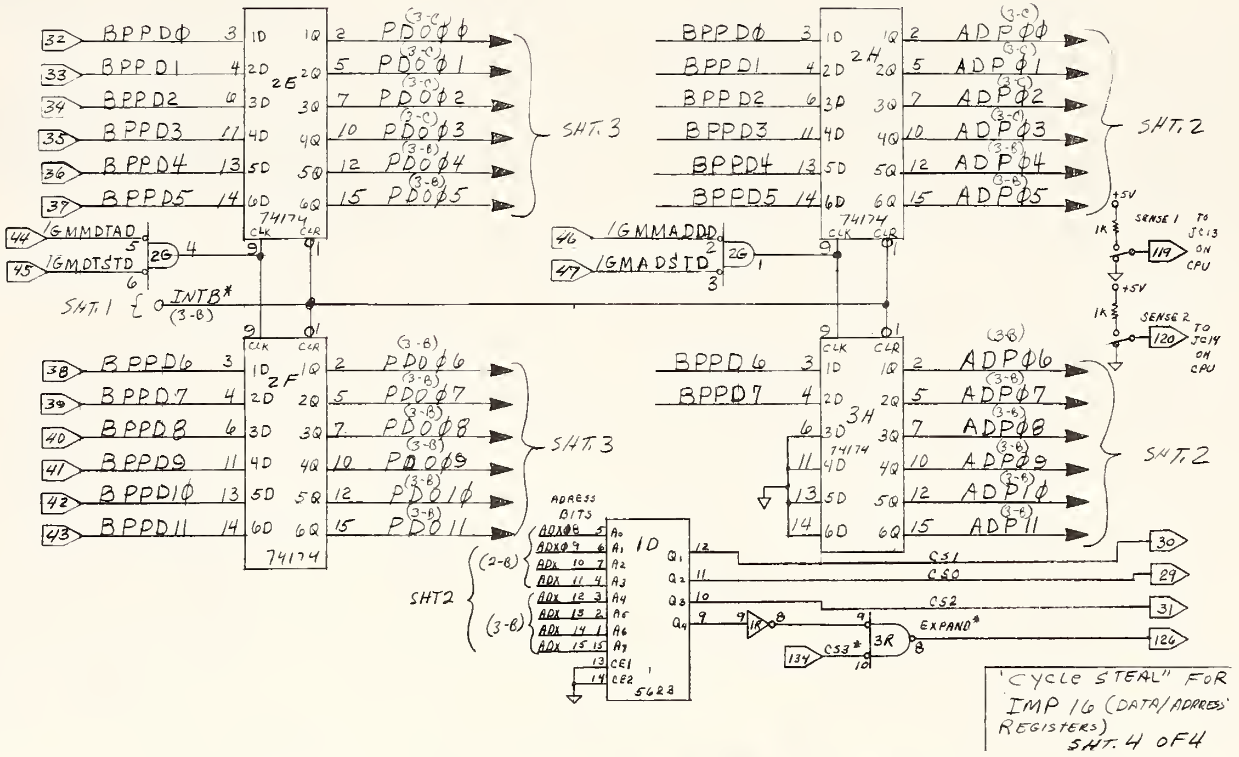
| | | | | |
|---|---|---|---|---|
| NEXT ASSY | USED ON | TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° ±1/2° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC.<br>SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
| APPLICATION | | MATERIAL | CONTRACT NO | "STEAL CIRCUIT" FOR IMP 16 (DATA BUSS) |
| | | FINISH–SEE NOTE | | SIZE C · CODE IDENT NO. 27327 · DRAWING NO. |
| | | | | SCALE · SHEET 3 OF 4 |

DF-8 REV. 7-75 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



'CYCLE STEAL" FOR IMP 16 (DATA/ADDRESS REGISTERS) SHT. 4 OF 4

| | |
|---|---|
| NEXT ASSY | USED ON |
| APPLICATION | |

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES +2° -1°
MATERIAL
FINISH–SEE NOTE

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

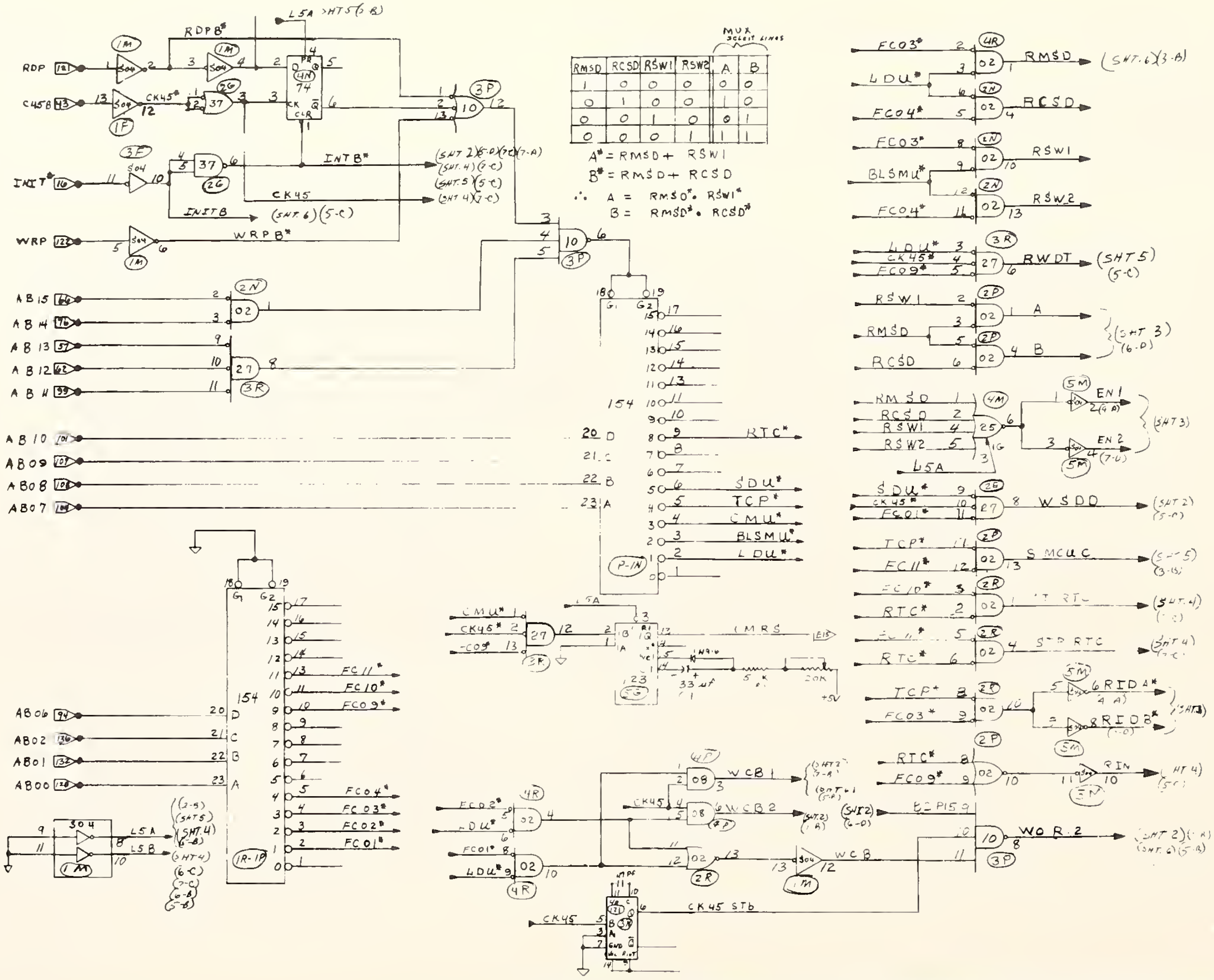
HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

"CYCLE STEAL" FOR IMP 16 (DATA/ADDRESS REGISTERS)

| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| C | 27327 | |

SCALE — SHEET 4 OF 4

DF-8 REV. 7-78 V680



| RMSD | RCSD | RSW1 | RSW2 | A | B |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |

A* = RMSD + RSW1

B* = RMSD + RCSD

∴ A = RMSD*• RSW1*

B = RMSD*• RCSD*

| TOLERANCES UNLESS NOTED OTHERWISE | | DRAFTSMAN | | |
|---|---|---|---|---|
| .X ± <br> .XX ± <br> .XXX ± | 90° FORMED ANGLES +2° −1° | CHECK | | |
| | | DEV ENGR | | |
| | | PROJ ENGR | | |
| | | RELIABILITY | | |
| FINISH — SEE NOTE | | CONTRACT NO. | | |
| MATERIAL | | | | |

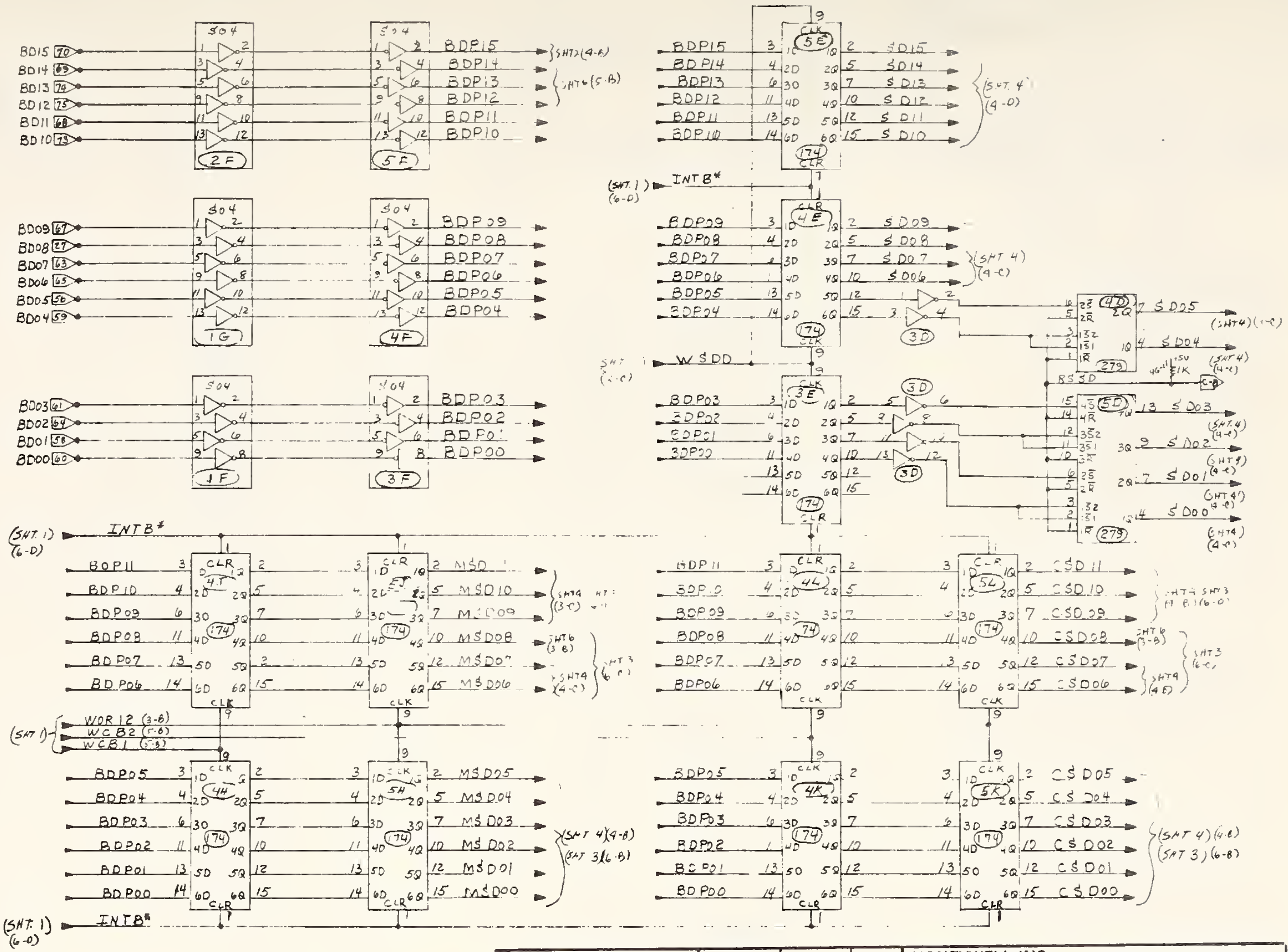
HONEYWELL INC. AEROSPACE AND DEFENSE GROUP

□ AERO MPLS □ AERO ST PETERSBURG □ ORD SEATTLE □ ORD HOPKINS □ ORD MONTGOMERYVILLE □ S&R CTR MPLS □ RADIATION CTR BOSTON

INPUT/OUTPUT CONTROL

SDMU

| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| D | | |
| SCALE | WT | SHEET 1 OF 6 |



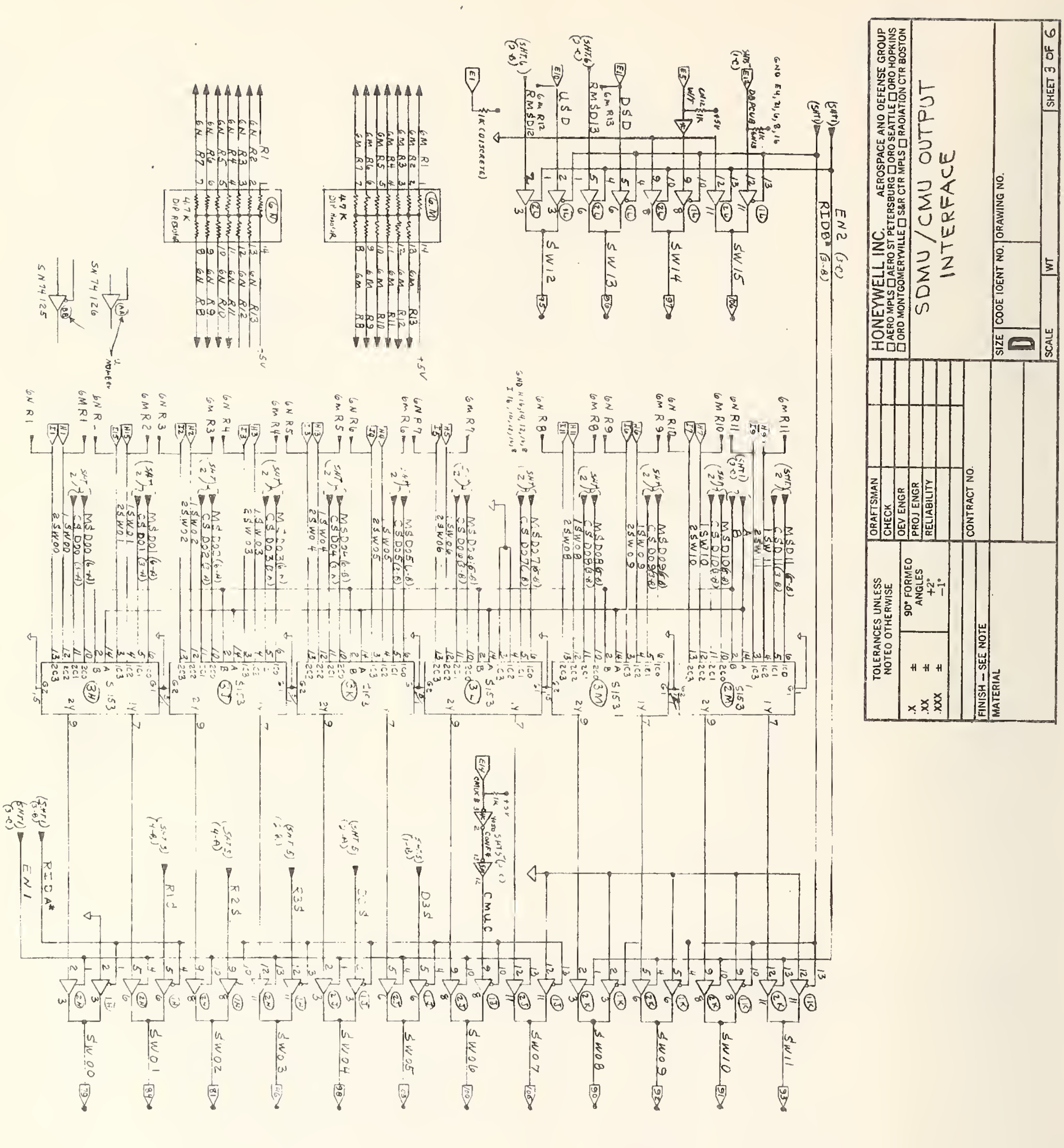
| TOLERANCES UNLESS NOTED OTHERWISE | | DRAFTSMAN | |
|---|---|---|---|
| .X ± <br> .XX ± <br> .XXX ± | 90° FORMED ANGLES +2° −1° | CHECK <br> DEV ENGR <br> PROJ ENGR <br> RELIABILITY | |
| | | CONTRACT NO. | |
| FINISH — SEE NOTE <br> MATERIAL | | | |

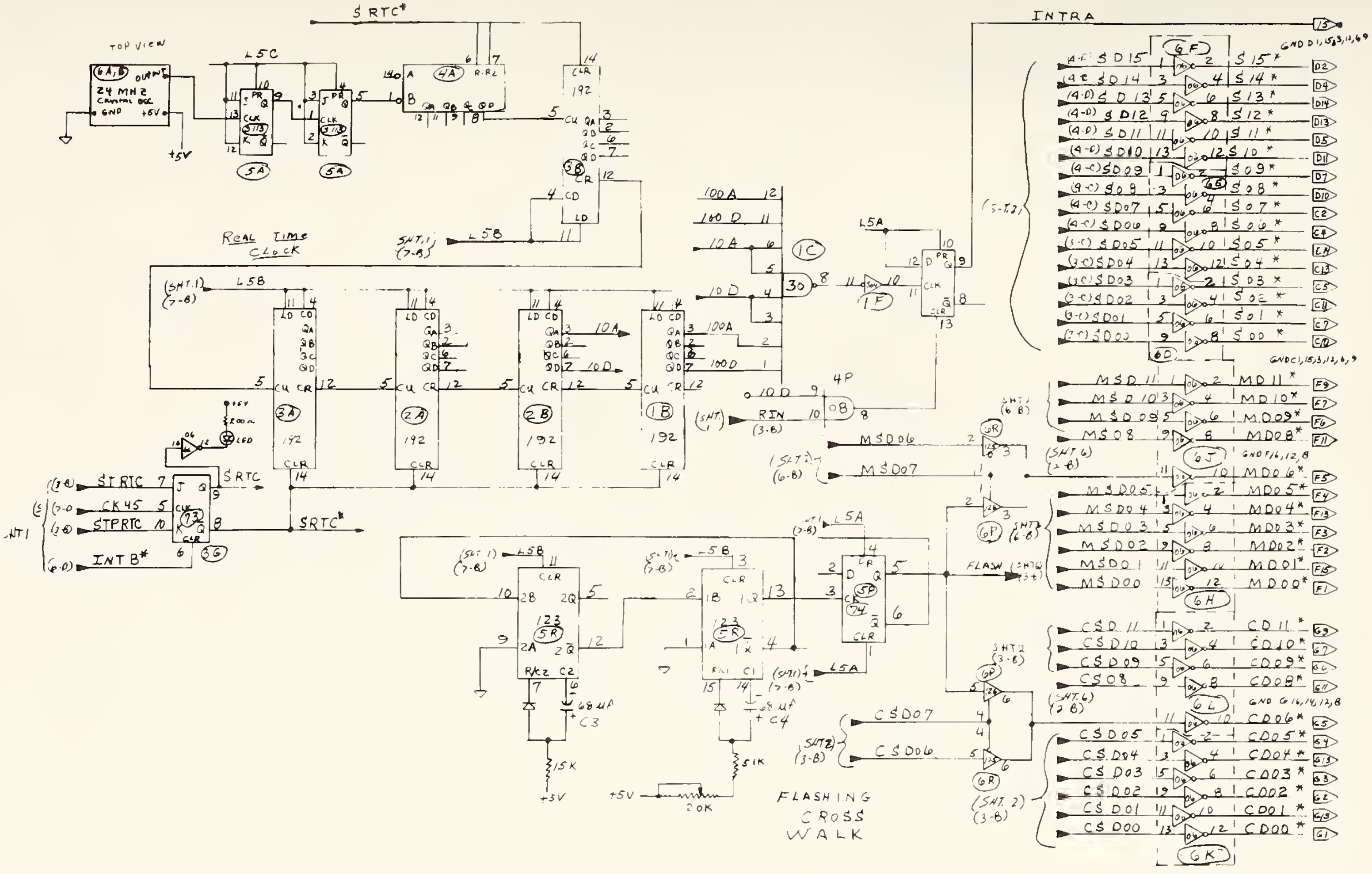
HONEYWELL INC. AEROSPACE AND DEFENSE GROUP
□ AERO MPLS □ AERO ST PETERSBURG □ ORO SEATTLE □ ORO HOPKINS
□ ORO MONTGOMERYVILLE □ S&R CTR MPLS □ RADIATION CTR BOSTON

SDMU/CMU
INPUT INTERFACE

| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| D | 27327 | |

SCALE | WT | SHEET 2 OF 6


HONEYWELL INC. AEROSPACE AND DEFENSE GROUP
□ AERO MPLS □ AERO ST PETERSBURG □ ORD SEATTLE □ ORD HOPKINS
□ ORD MONTGOMERYVILLE □ S&R CTR MPLS □ RADIATION CTR BOSTON
SDMU/CMU OUTPUT INTERFACE
TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES +2° −1°
FINISH — SEE NOTE
MATERIAL
DRAFTSMAN
CHECK
DEV ENGR
PROJ ENGR
RELIABILITY
CONTRACT NO.
SIZE D
CODE IDENT NO.
DRAWING NO.
SCALE
WT
SHEET 3 OF 6
EN2 (3-C)
RIDB* (3-B)
SW15
SW14
SW13
SW12
DSD
USD
RMSD13
RMSD12
DBPCUB
W/T
1K (DISCRETE)
GND E4, 21, 6, 8, 16
47K DIP RESISTOR
6M
6N
+5V
SN74126
SN74125
S153
MSD00
CSD00
1SW00
2SW00
MSD11
CSD11
1SW11
2SW11
CMUC
D3S
R3S
R2S
R1S
RIDA*
EN1
SW00
SW01
SW02
SW03
SW04
SW05
SW06
SW07
SW08
SW09
SW10
SW11



TOLERANCES UNLESS NOTED OTHERWISE

.X ±
.XX ±
.XXX ±

90° FORMED ANGLES +2° −1°

FINISH — SEE NOTE

MATERIAL

DRAFTSMAN

CHECK

DEV ENGR

PROJ ENGR

RELIABILITY

CONTRACT NO.

HONEYWELL INC. AEROSPACE AND DEFENSE GROUP

□ AERD MPLS □ AERO ST PETERSBURG □ ORD SEATTLE □ ORD HOPKINS
□ ORD MONTGOMERYVILLE □ S&R CTR MPLS □ RADIATION CTR BOSTON

SMCU — PERIPHERAL OUTPUT INTERFACE — FLASHING — CROSSWALK-RTC

SIZE D

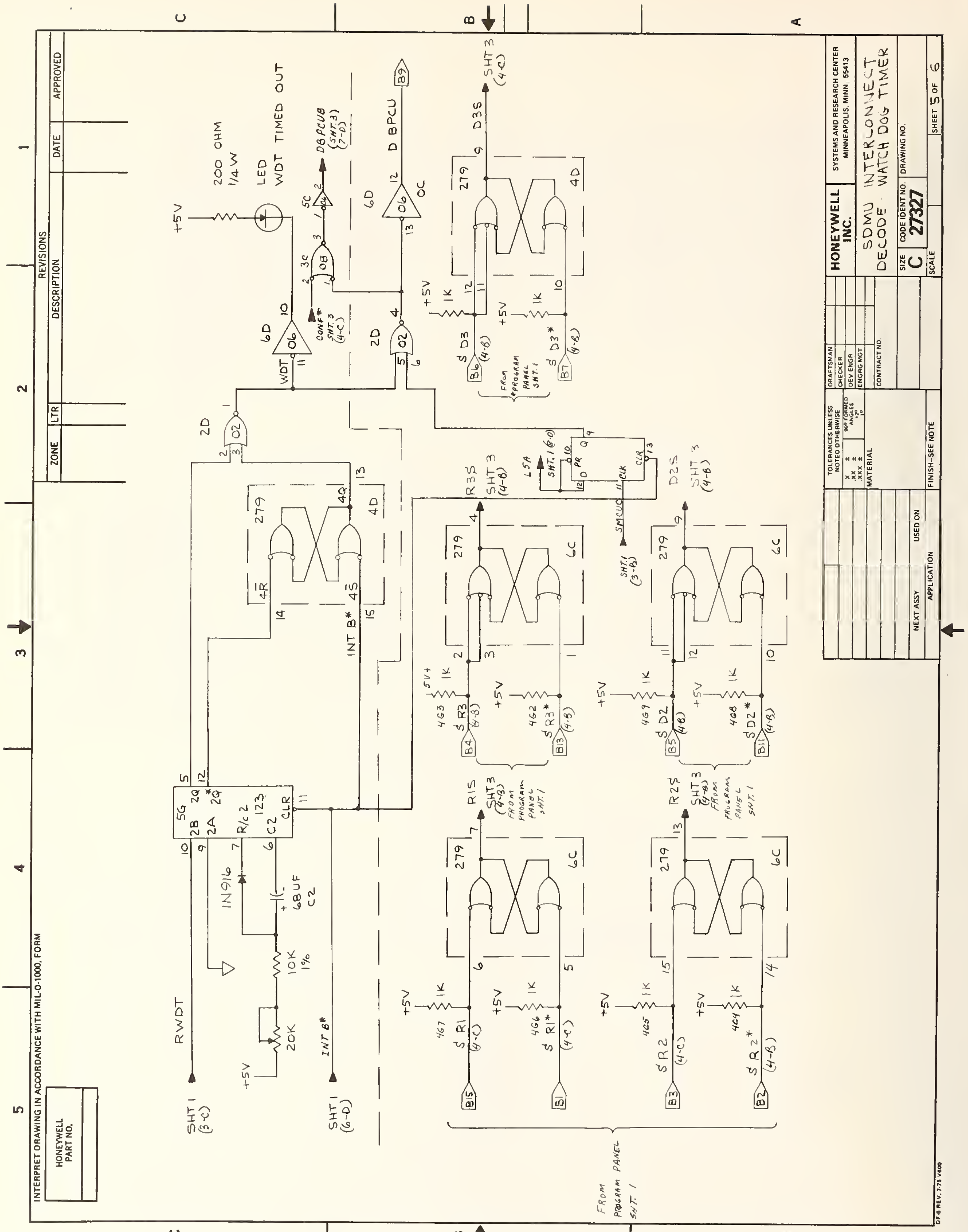
CODE IDENT NO.

DRAWING NO.

SCALE

WT

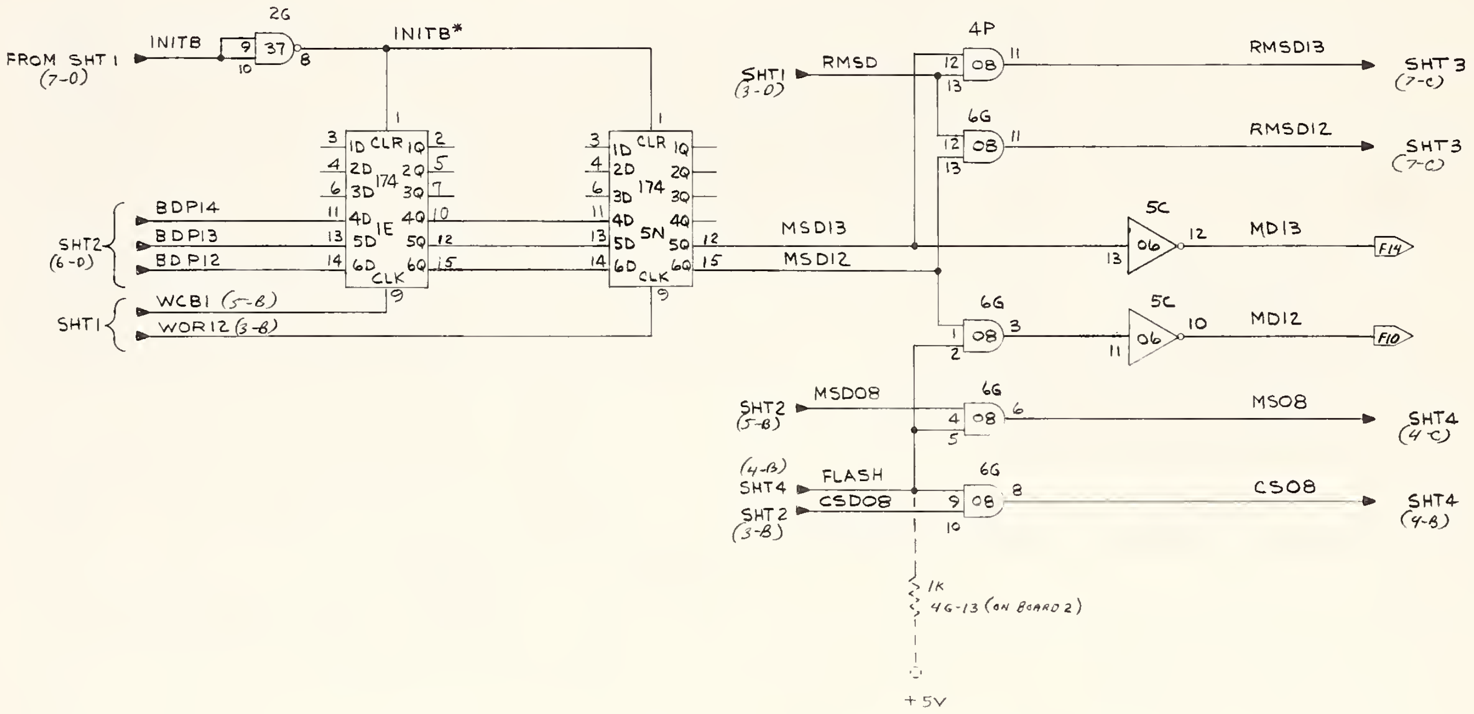
SHEET 4 OF 6


INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM
HONEYWELL PART NO.
REVISIONS
ZONE
LTR
DESCRIPTION
DATE
APPROVED
HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN 55413
SDMU INTERCONNECT DECODE WATCH DOG TIMER
SIZE C
CODE IDENT NO. 27327
DRAWING NO.
SCALE
SHEET 5 OF 6
TOLERANCES UNLESS NOTED OTHERWISE
MATERIAL
FINISH–SEE NOTE
NEXT ASSY
USED ON
APPLICATION
DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.
RWDT
SHT 1 (3-C)
+5V
20K
10K 1%
1N916
68UF C2
INT B*
SHT 1 (6-D)
5G
2B
2A
R/C 2
C2
2Q
2Q*
123
CLR
279
4R
4S
4Q
4D
2D
02
INT B*
WDT
6D
06
CONF* SHT. 3 (4-C)
3C
08
5C
DBPCUB (SHT 3) (7-D)
+5V
200 OHM 1/4 W
LED WDT TIMED OUT
D BPCU
B9
0C
SHT 3 (4-C)
D3S
$ D3 (4-B)
$ D3* (4-B)
B6
B7
FROM PROGRAM PANEL SHT. 1
1K
PR
CLK
D
Q
L5A
SHT. 1 (6-D)
SMCUC
SHT. 1 (3-B)
R3S
SHT 3 (4-B)
D2S
SHT 3 (4-B)
6C
$ R3 (4-B)
$ R3* (4-B)
$ D2 (4-B)
$ D2* (4-B)
B4
B13
B5
B11
R1S
SHT 3 (4-B) FROM PROGRAM PANEL SHT. 1
R2S
SHT 3 (4-B) FROM PROGRAM PANEL SHT. 1
$ R1 (4-C)
$ R1* (4-C)
$ R2 (4-C)
$ R2* (4-B)
B15
B1
B3
B2
FROM PROGRAM PANEL SHT. 1
DF-8 REV. 7-78 V800

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



NOTE:
REMOVE "FLASH" FROM PINS 9,5 & 2 OF 6G
AND TIE THEM UP TO +5V VOLTS THROUGH A
1K RESISTOR IN THE MPLS SYSTEM

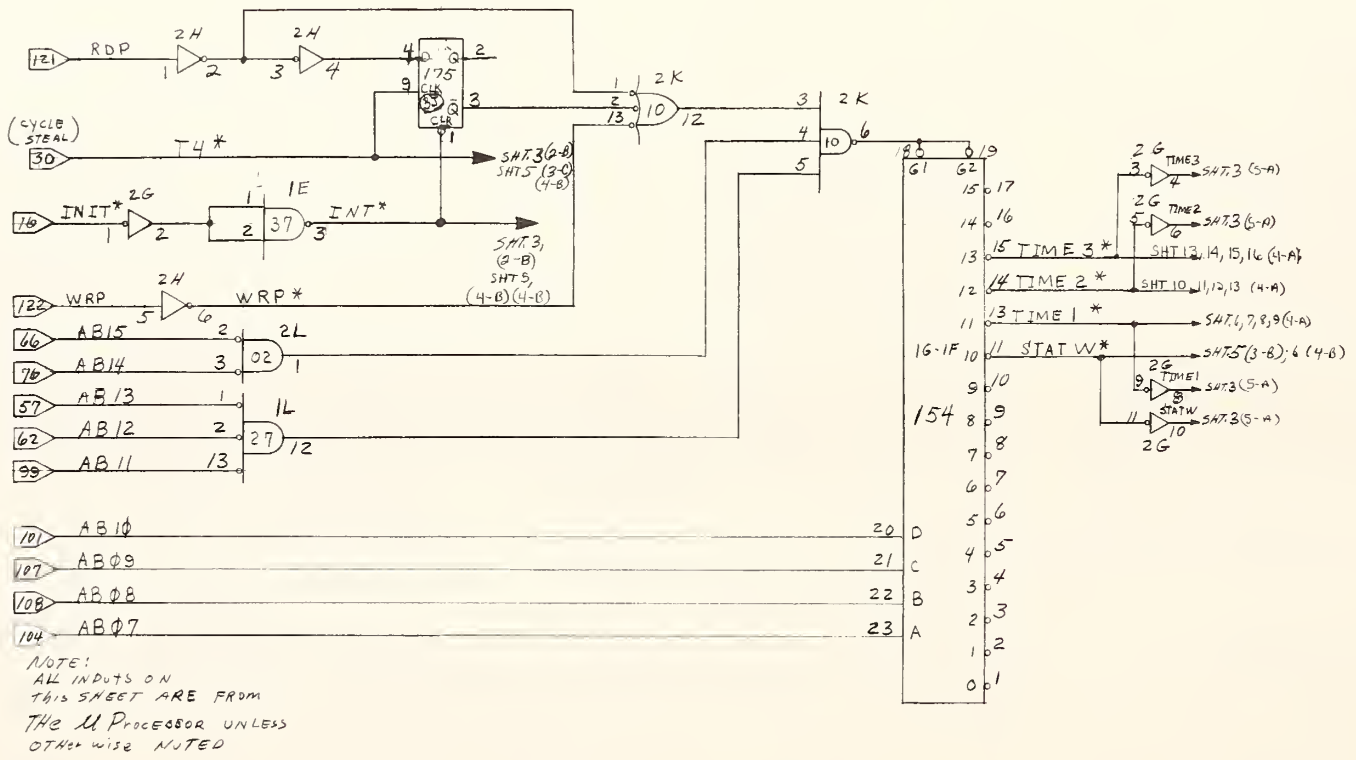
| | | | |
|---|---|---|---|
| TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2°<br>1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
| MATERIAL | CONTRACT NO | SDMU-MODIFICATION<br>CIRCUITS FOR WASHINGTON | |
| NEXT ASSY / USED ON | | SIZE C | CODE IDENT NO. 27327 / DRAWING NO. |
| APPLICATION | FINISH—SEE NOTE | SCALE | SHEET 6 OF 6 |

DF-8 REV. 7-75 V600

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. |
|---|
| |

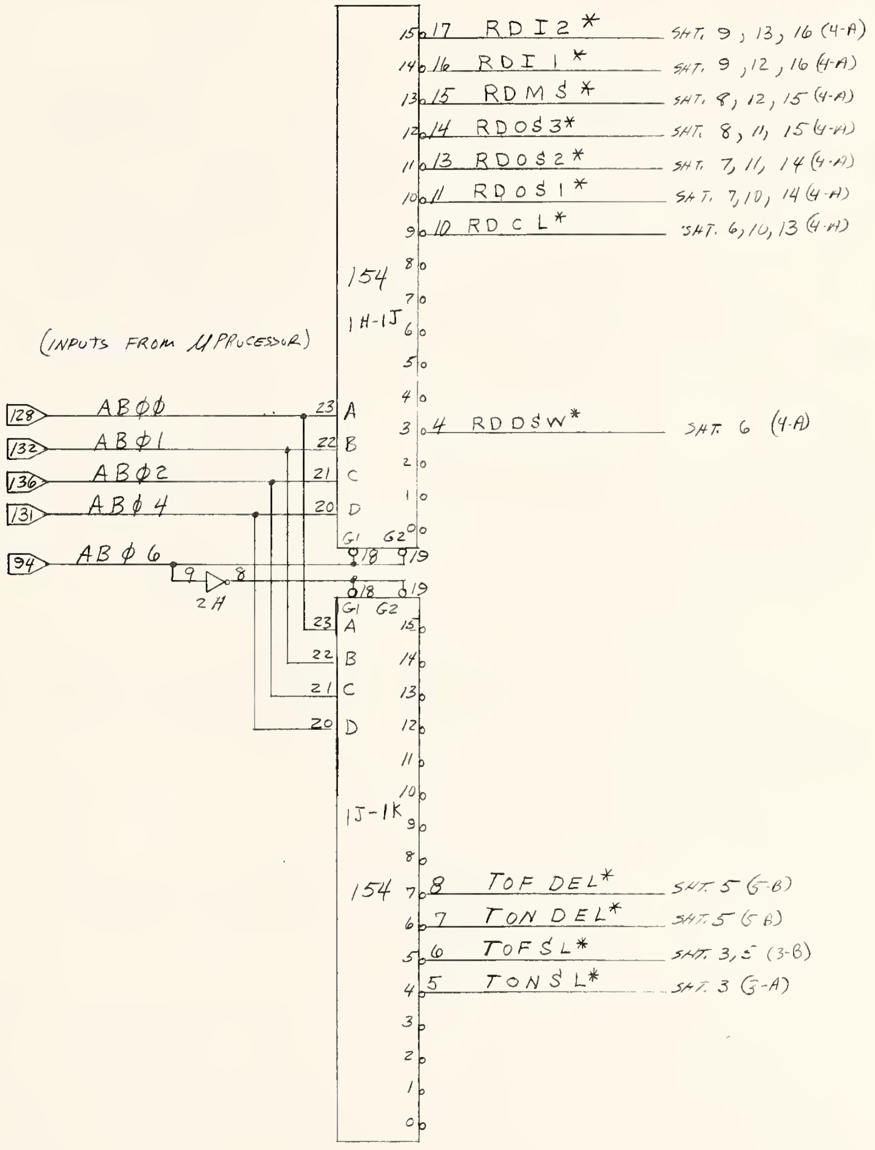
| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



NOTE:
ALL INPUTS ON
THIS SHEET ARE FROM
THE μ PROCESSOR UNLESS
OTHERWISE NOTED

| | | | |
|---|---|---|---|
| TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° ±1° | DRAFTSMAN<br>CHECKER<br>DEV ENGR<br>ENGRG MGT | HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
| MATERIAL | CONTRACT NO | PANEL ELECTRONICS<br>PERIPHERAL DECODE | |
| NEXT ASSY | USED ON | SIZE C | CODE IDENT NO. 27327 | DRAWING NO. |
| APPLICATION | FINISH–SEE NOTE | SCALE | SHEET 1 OF 16 |

DF-8 REV. 7-75 V600



REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

| NEXT ASSY | USED ON |
|---|---|
| APPLICATION | |

TOLERANCES UNLESS NOTED OTHERWISE
X ±
XX ±
XXX ±
90° FORMED ANGLES ±1°
MATERIAL
FINISH–SEE NOTE

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

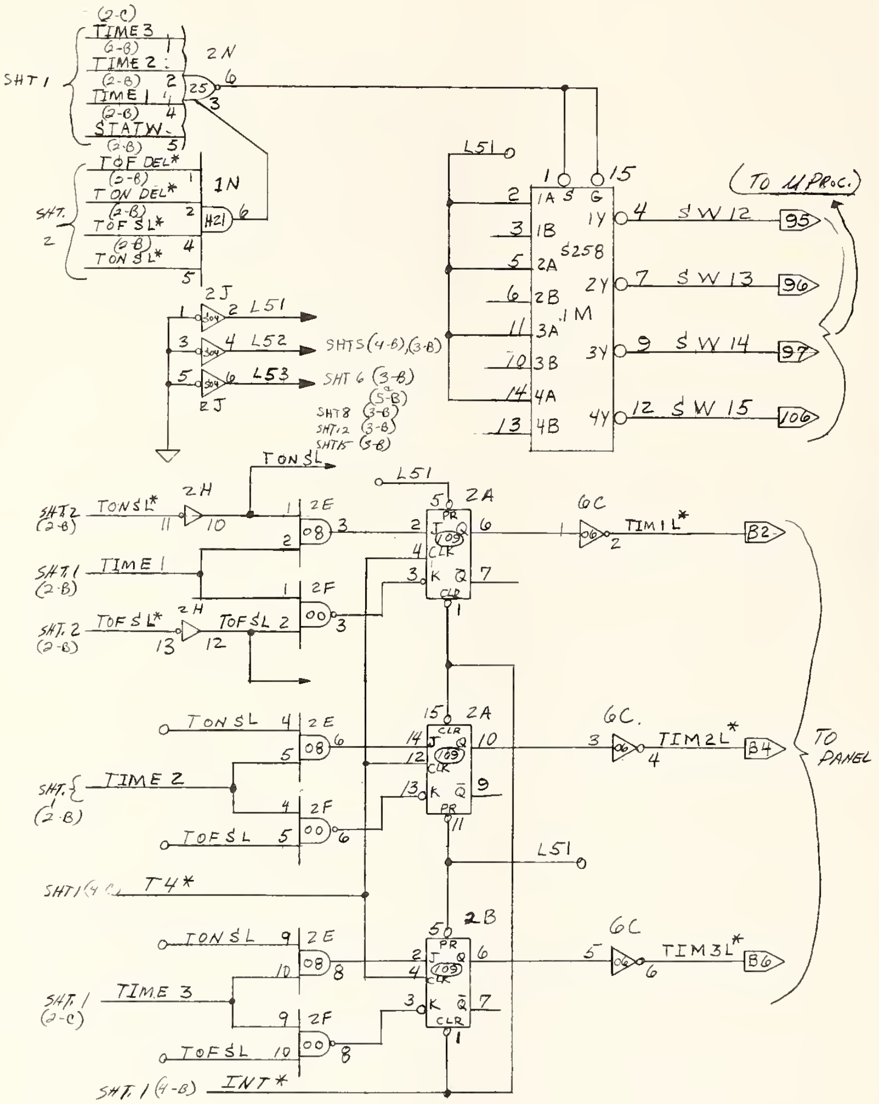
HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PANEL ELECTRONICS FUNCTION DECODE

SIZE C | CODE IDENT NO. 27327 | DRAWING NO.
SCALE | SHEET 2 OF 16

DF-8 REV. 7-78 V600

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | REVISIONS | | |



| | |
|---|---|
| HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
| PANEL ELECTRONICS<br>MSB'S OUTPUT MUX &<br>ERROR LIGHT CIRCUIT | |
| SIZE C | CODE IDENT NO. 27327 | DRAWING NO. |
| SCALE | SHEET 3 OF 16 |

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

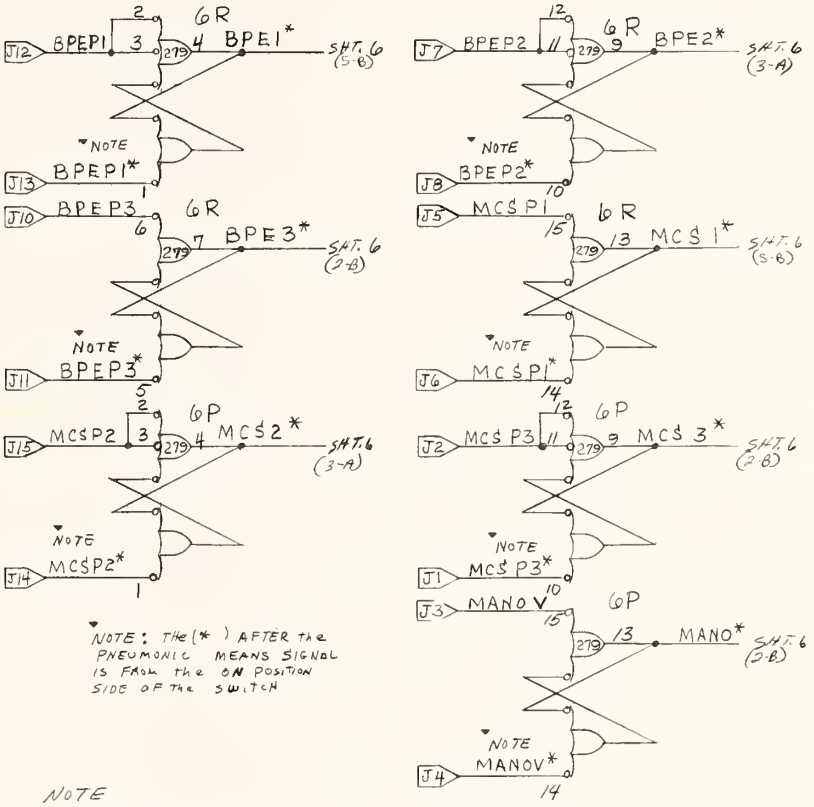
TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2°
MATERIAL
FINISH–SEE NOTE

NEXT ASSY | USED ON
APPLICATION

INTERPRET DRAWING IN ACCORDANCE WITH MIL-O-1000, FORM

HONEYWELL PART NO.

DF-8 REV. 7-78 V600



NOTE: THE (* ) AFTER the PNEUMONIC MEANS SIGNAL IS FROM the ON POSITION SIDE OF the SWITCH

NOTE
ALL INPUTS ON THIS SHEET ARE FROM THE PANEL

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FDRM

HONEYWELL PART NO.

| | |
|---|---|
| HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER MINNEAPOLIS, MINN. 55413 |
| PANEL ELECTRONICS STATUS WORD LATCH'S | |
| SIZE C | CODE IDENT NO. 27327 |
| DRAWING NO. | |
| SCALE | SHEET 4 OF 16 |

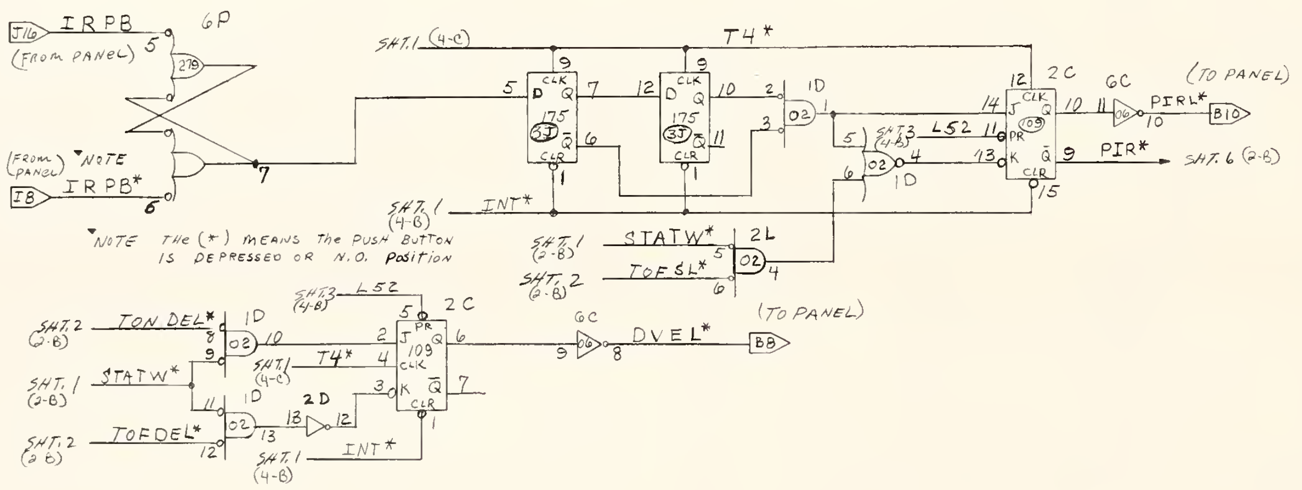
DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.
TOLERANCES UNLESS NOTED OTHERWISE
MATERIAL
FINISH—SEE NOTE
NEXT ASSY
USED ON
APPLICATION

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |



*NOTE THE (*) MEANS THE PUSH BUTTON IS DEPRESSED OR N.O. POSITION

| | |
|---|---|
| NEXT ASSY | USED ON |
| APPLICATION | |

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES +2° -1°
MATERIAL
FINISH—SEE NOTE

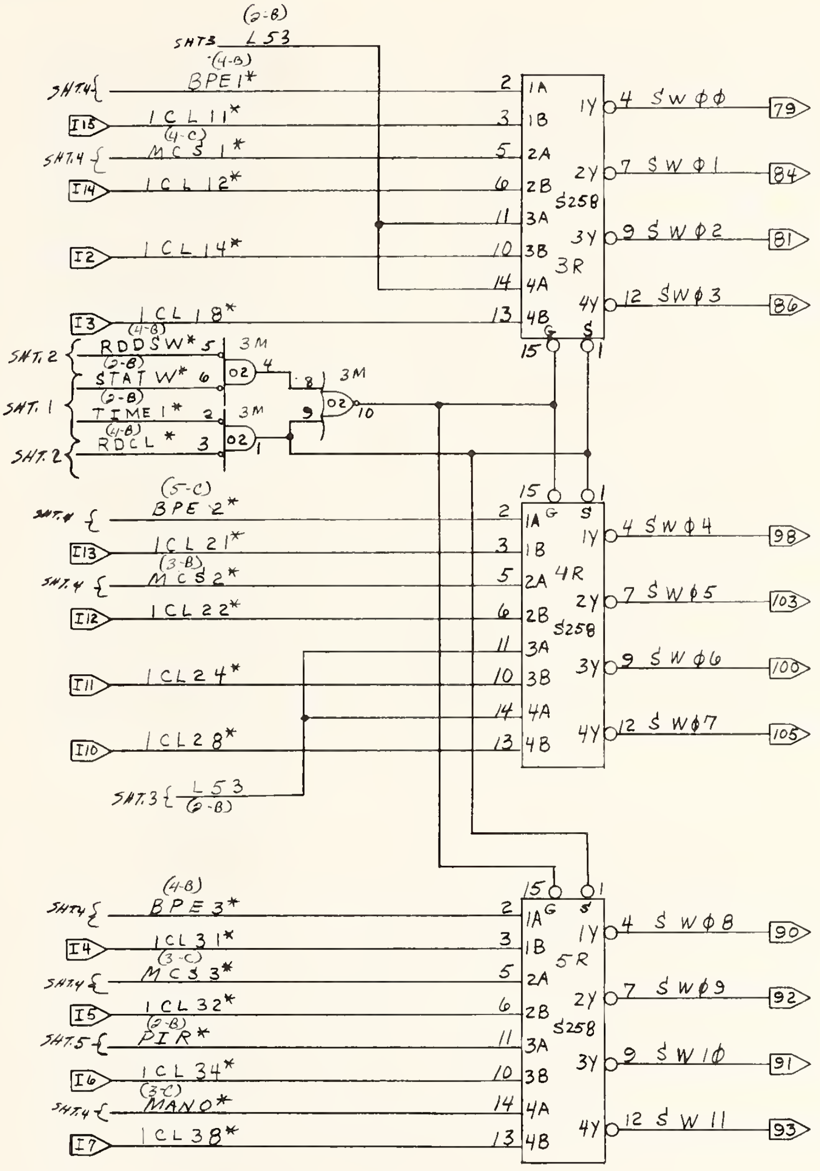
DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO

HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PANEL ELECTRONICS
INPUT REQUEST/ DATA
VALIDITY

SIZE C
CODE IDENT NO. 27327
DRAWING NO.
SCALE
SHEET 5 OF 16

DF-6 REV. 7-75 V600



NOTE: INPUT PINS COME FROM PANEL
OUTPUT PINS GO TO µ PROCESSOR

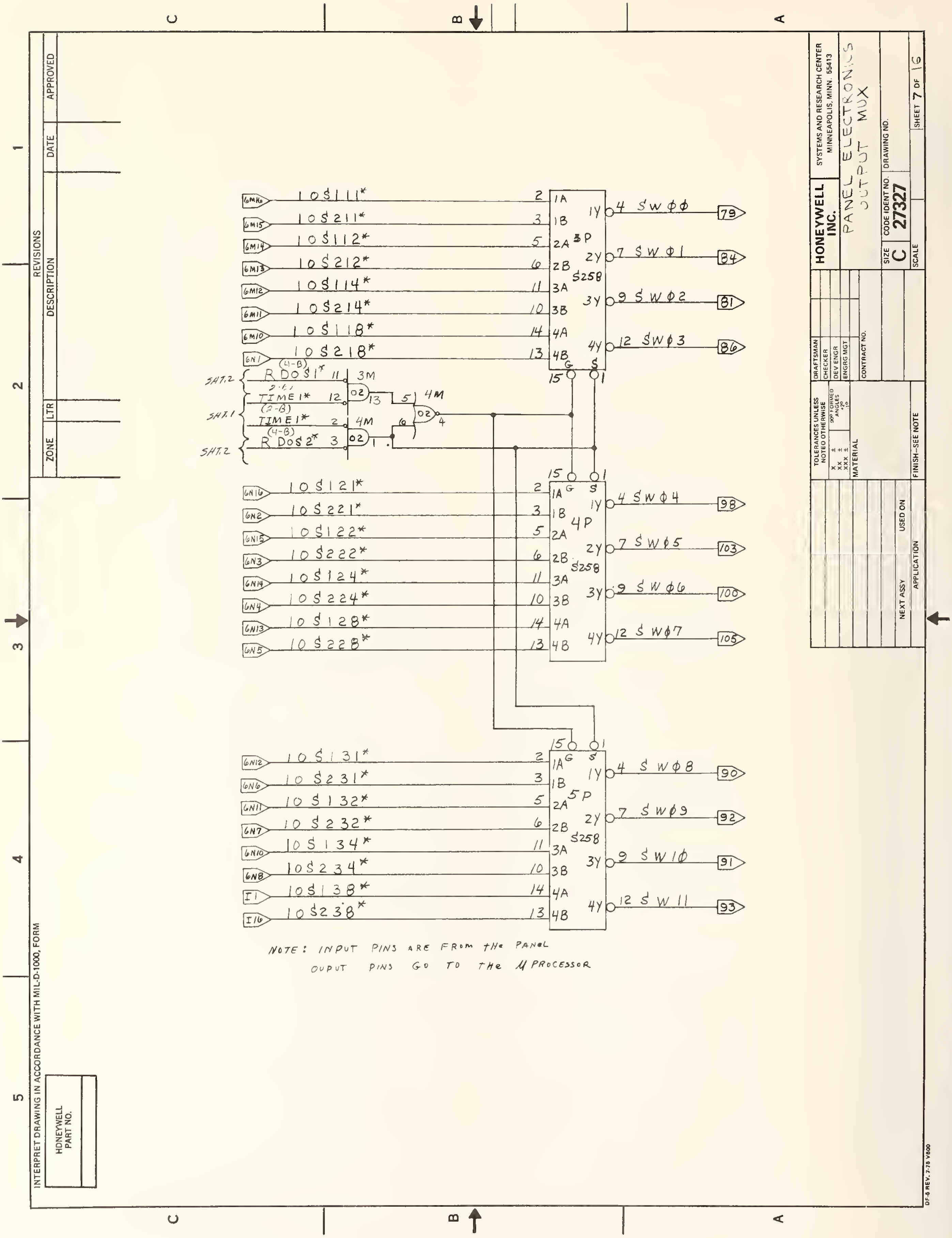
HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PANEL ELECTRONICS
OUTPUT MUX

SIZE C
CODE IDENT NO. 27327
DRAWING NO.
SCALE
SHEET 6 OF 16

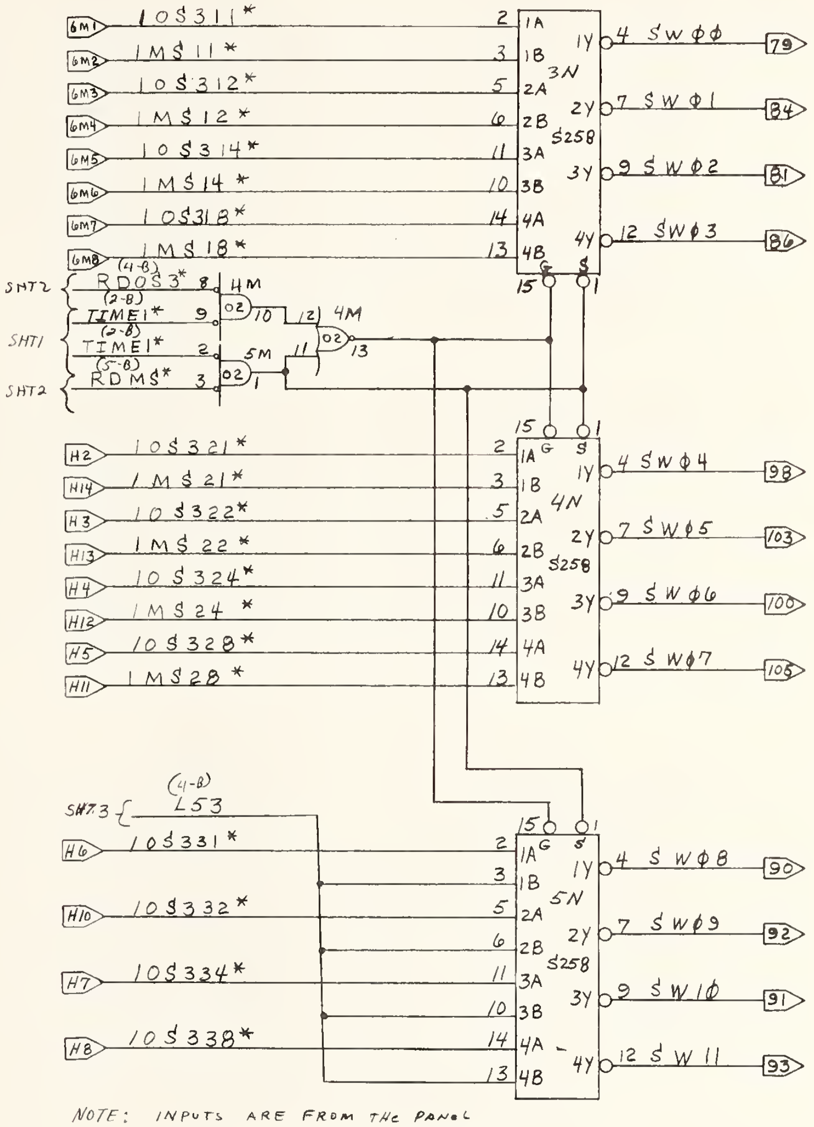
INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.



NOTE: INPUT PINS ARE FROM THE PANEL
OUPUT PINS GO TO THE μPROCESSOR

| ZONE | LTR | REVISIONS<br>DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |



NOTE: INPUTS ARE FROM THE PANEL
OUTPUTS ARE TO THE μ PROCESSOR

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

TOLERANCES UNLESS NOTED OTHERWISE
X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2°
ANGLES ±1°
MATERIAL

FINISH–SEE NOTE

| NEXT ASSY | USED ON |
|---|---|
| APPLICATION | |

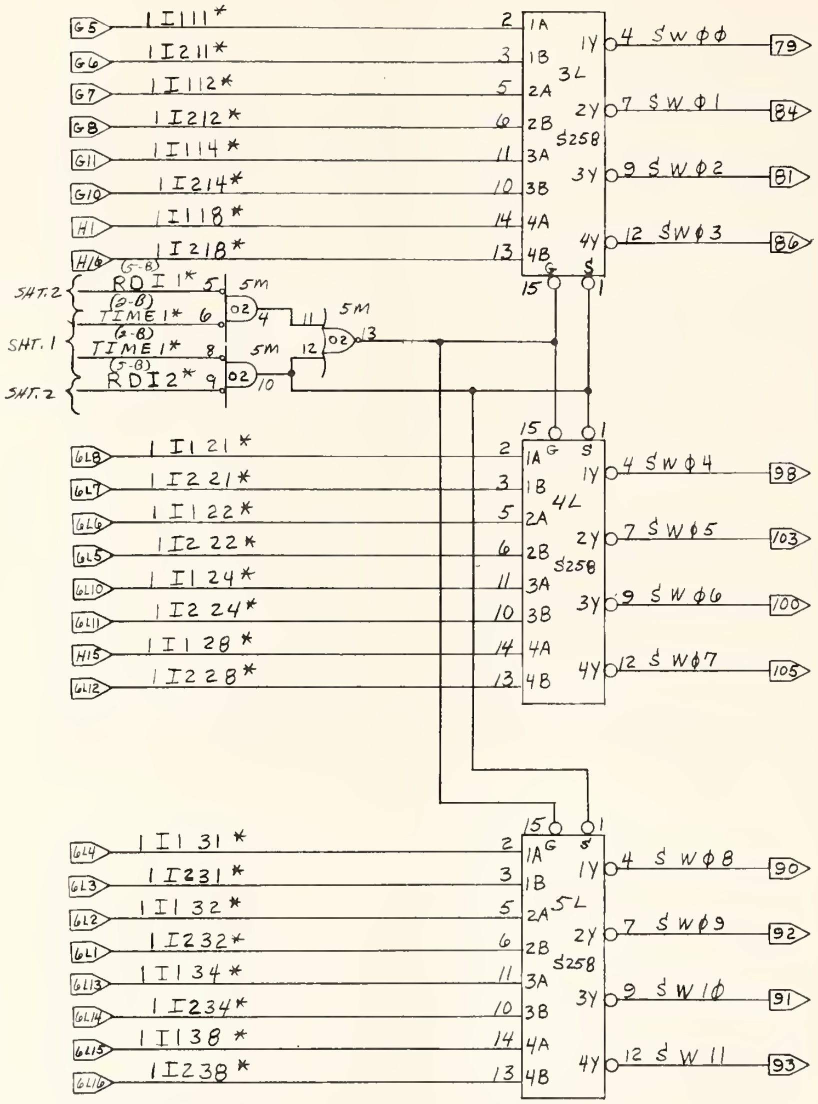
DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PANEL ELECTRONICS
OUTPUT MUX

SIZE C
CODE IDENT NO. 27327
DRAWING NO.
SCALE
SHEET 8 OF 16

DF-5 REV. 7-75 V600



NOTE: INPUTS ARE FROM THE PANEL
OUTPUTS ARE TO THE μ PROCESSOR

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|

HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN 55413

PANEL ELECTRONICS
OUTPUT MUX

SIZE C | CODE IDENT NO. 27327 | DRAWING NO.
SCALE | SHEET 9 OF 16

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2° ±1°
MATERIAL
FINISH–SEE NOTE

NEXT ASSY | USED ON
APPLICATION

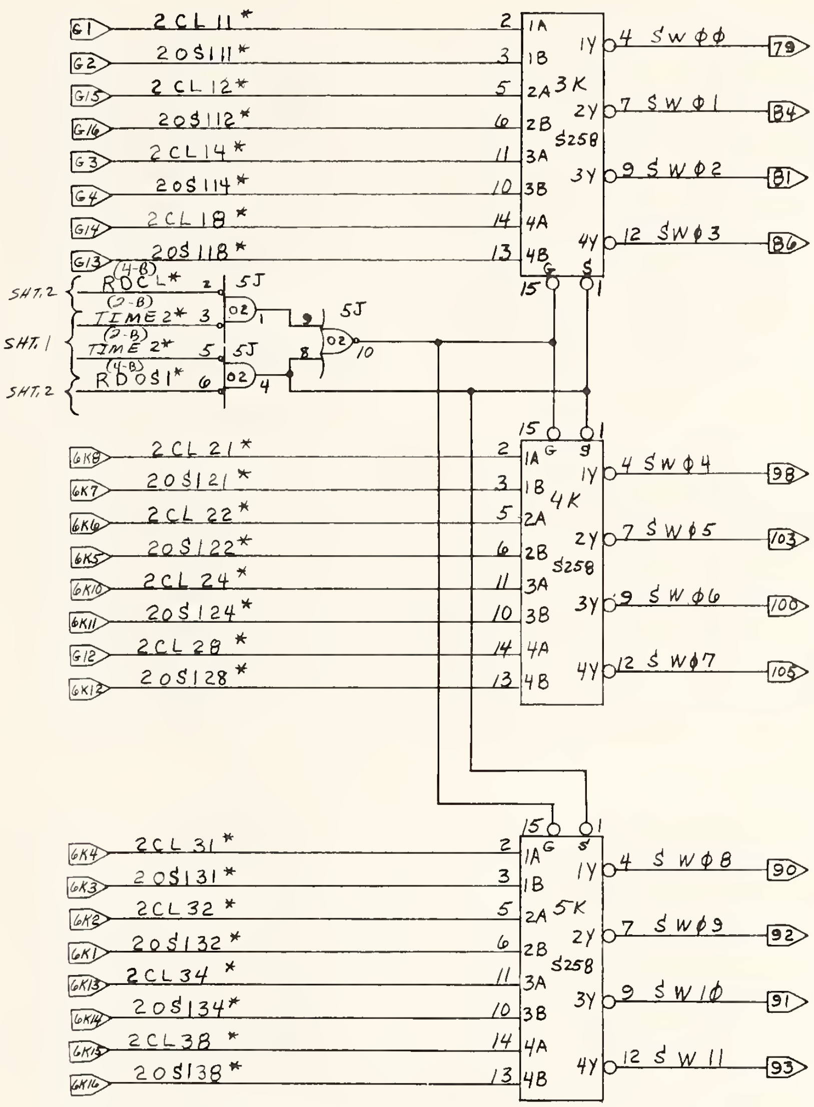
HONEYWELL PART NO.

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

DF-8 REV. 7-78 V600

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|



NOTE: INPUTS ARE FROM the PANEL
OUTPUTS ARE TO The µ PROCESSOR

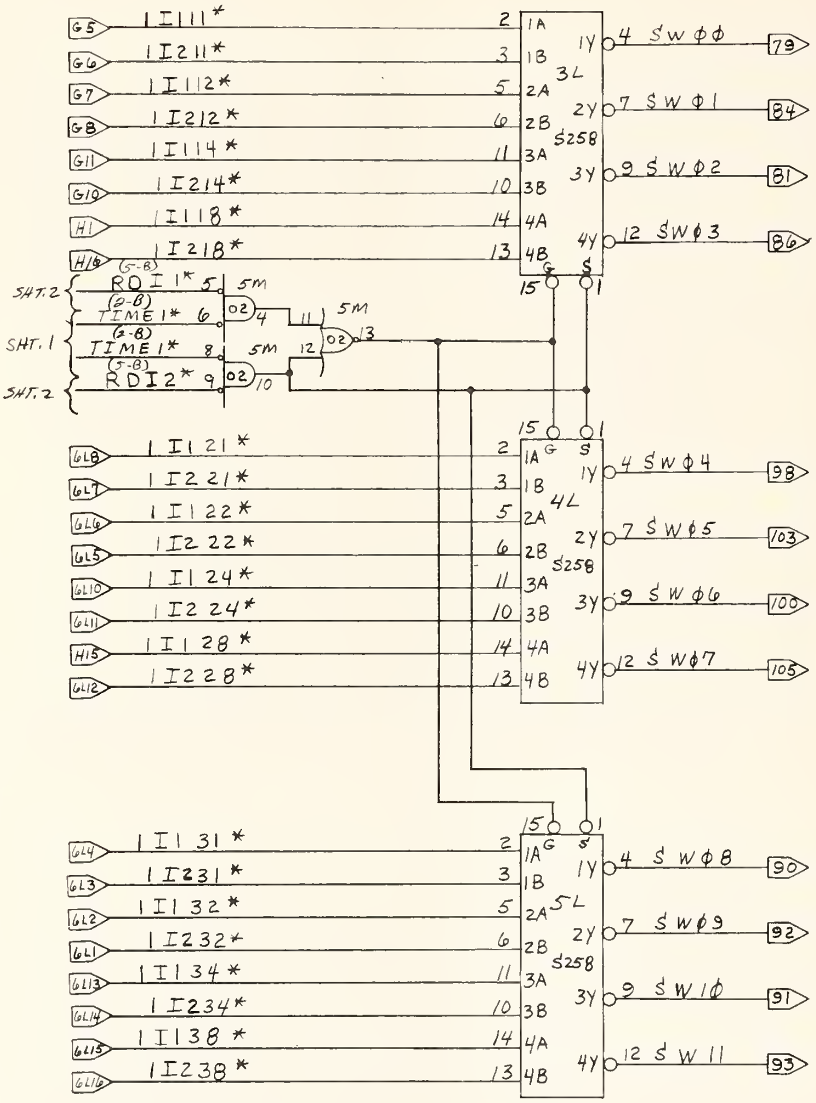
| | |
|---|---|
| HONEYWELL INC. | SYSTEMS AND RESEARCH CENTER<br>MINNEAPOLIS, MINN. 55413 |
| DRAFTSMAN | |
| CHECKER | |
| DEV ENGR | |
| ENGRG MGT | |
| CONTRACT NO | PANEL ELECTRONICS<br>OUTPUT MUX |
| TOLERANCES UNLESS NOTED OTHERWISE<br>.X ±<br>.XX ±<br>.XXX ±<br>90° FORMED ANGLES ±2° ±1° | SIZE C &#124; CODE IDENT NO. 27327 &#124; DRAWING NO. |
| MATERIAL | |
| FINISH–SEE NOTE | SCALE &#124; SHEET 10 OF 16 |
| NEXT ASSY &#124; USED ON | |
| APPLICATION | |

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| HONEYWELL PART NO. | |
|---|---|

DF-5 REV. 7-75 V600

| ZONE | LTR | REVISIONS<br>DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |



NOTE: INPUTS ARE FROM THE PANEL
OUTPUTS ARE TO THE μ PROCESSOR

HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PANEL ELECTRONICS
OUTPUT MUX

| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| C | 27327 | |

SCALE

SHEET 9 OF 16

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO

TOLERANCES UNLESS NOTED OTHERWISE
X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2° ±10

MATERIAL

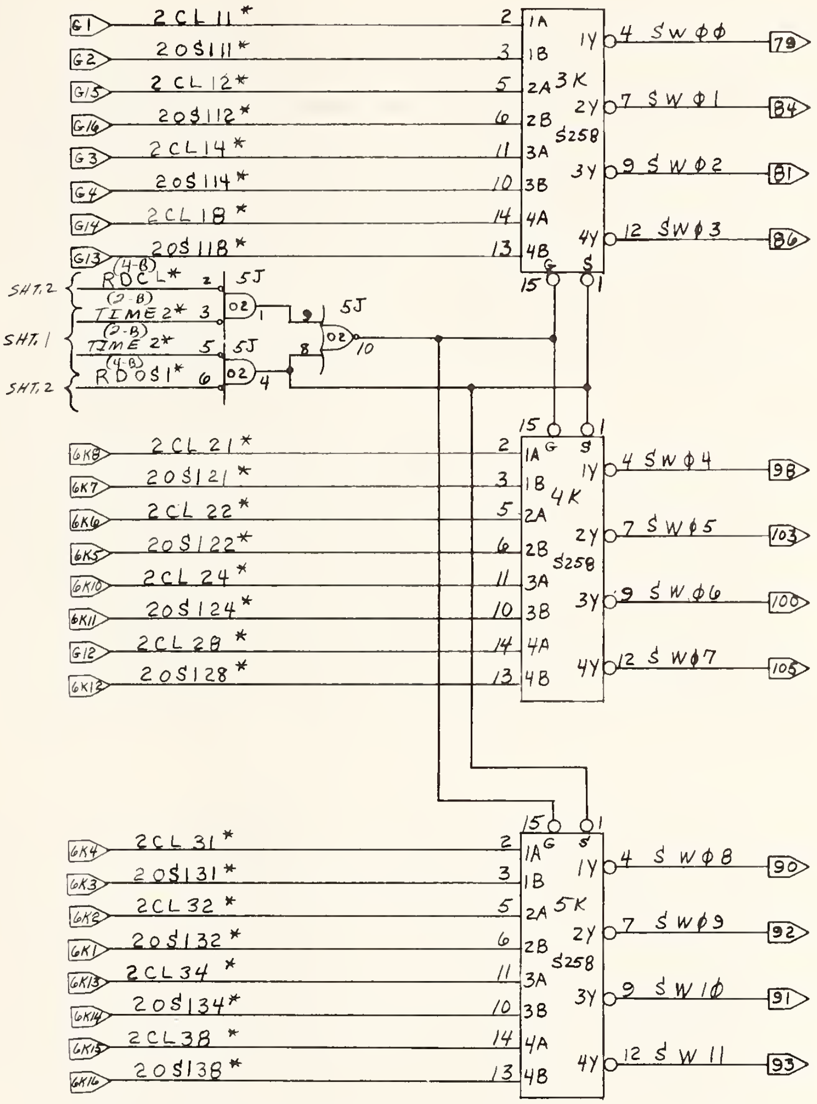
FINISH–SEE NOTE

NEXT ASSY | USED ON
APPLICATION

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

DF-8 REV. 7-78 V600



NOTE: INPUTS ARE FROM The PANEL
OUTPUTS ARE TO The μ PROCESSOR

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|

REVISIONS

HONEYWELL INC.

SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PANEL ELECTRONICS
OUTPUT MUX

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2°
MATERIAL
FINISH-SEE NOTE

NEXT ASSY | USED ON
APPLICATION

SIZE C | CODE IDENT NO. 27327 | DRAWING NO.
SCALE | SHEET 10 OF 16

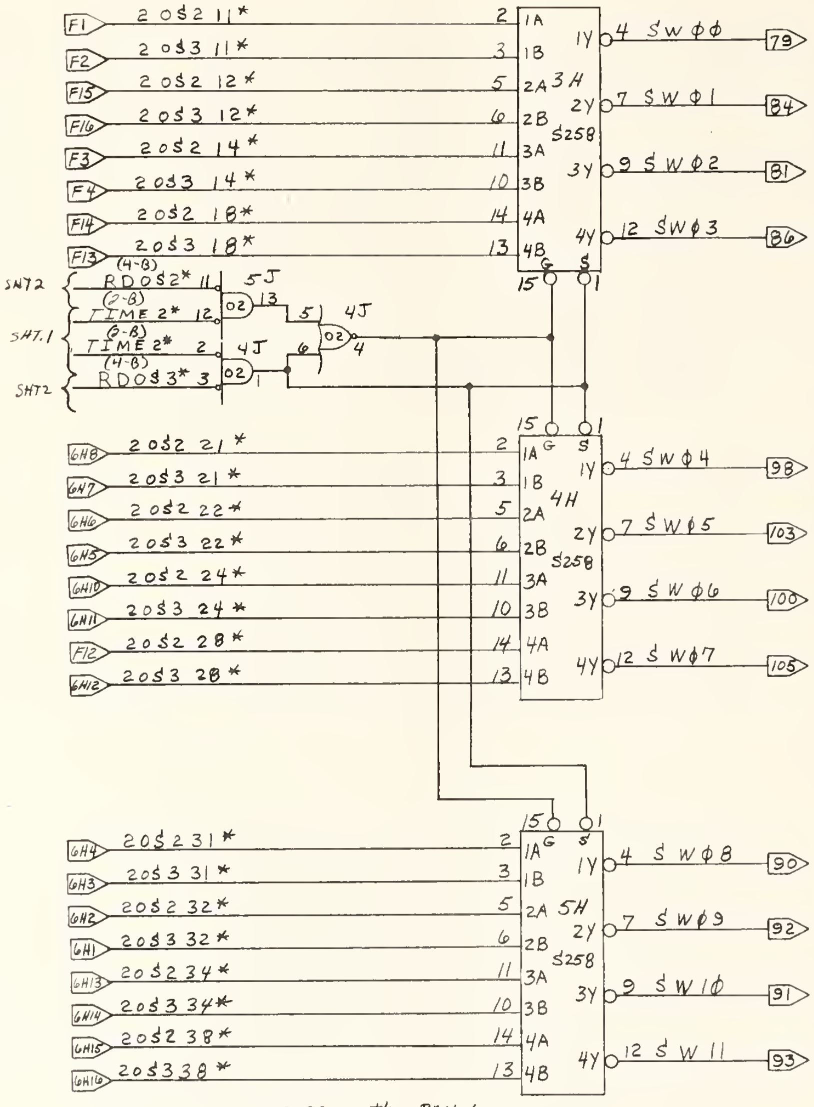
HONEYWELL PART NO.

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

DF-8 REV. 7-75 V600

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |



NOTE! IPuts ARE FROM the PANeL
OUTPuts ARE TO the μPROCESSOR

HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PANEL ELECTRONICS
OUTPUT MUX

| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| C | 27327 | |

SCALE SHEET 11 OF 16

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

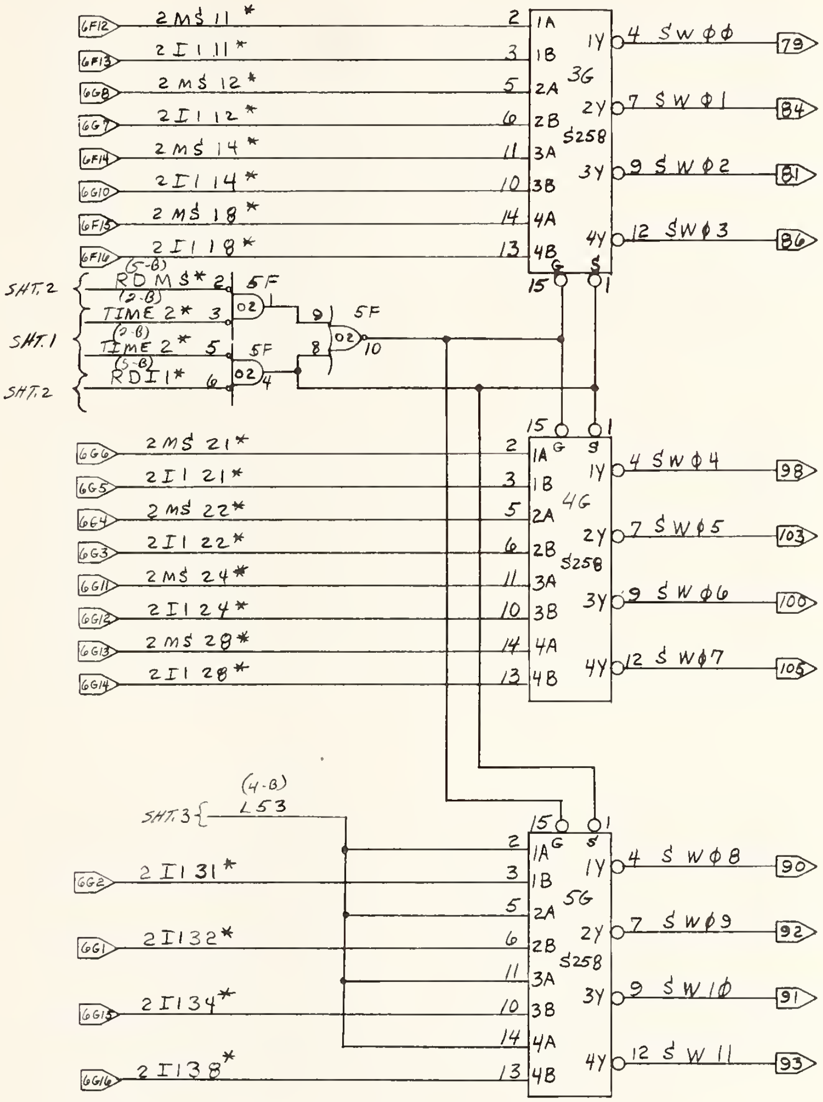
TOLERANCES UNLESS NOTED OTHERWISE
X ±
XX ±
XXX ±
90° FORMED ANGLES ±2°
±1°
MATERIAL
FINISH–SEE NOTE

NEXT ASSY | USED ON
APPLICATION

INTERPRET DRAWING IN ACCORDANCE WITH MIL-0-1000, FORM

HONEYWELL PART NO.

DF-8 REV. 7-75 V800



NOTE! INPUT PINS ARE FROM the PANEL
OUTPUT PINS ARE TO the µ PROCESSOR

REVISIONS

| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

HONEYWELL INC.

SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PANEL ELECTRONICS
OUTPUT MUX

| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| C | 27327 | |

SCALE

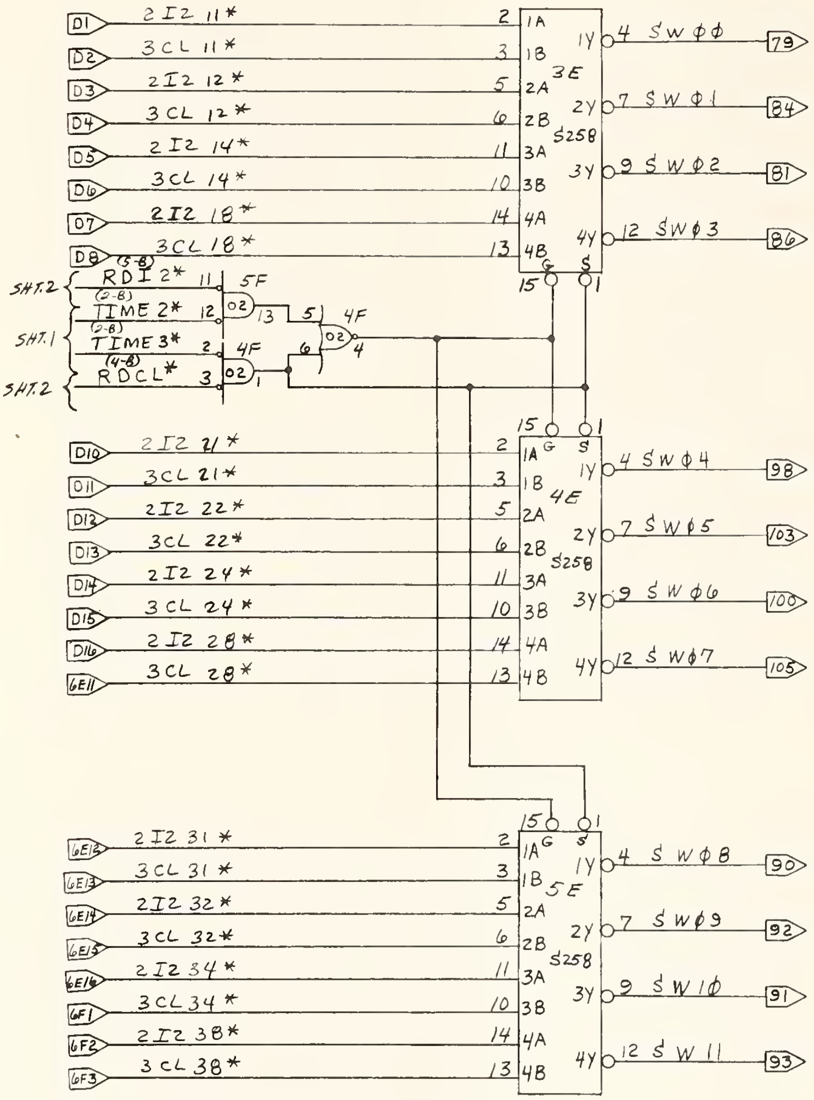
SHEET 12 OF 16

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

TOLERANCES UNLESS NOTED OTHERWISE
X ±
XX ±
XXX ±
90° FORMED ANGLES ±2° ±1°
MATERIAL
FINISH—SEE NOTE

NEXT ASSY | USED ON
APPLICATION

DF-5 REV. 7-75 V600



NOTE: INPUTS ARE FROM THE PANEL
OUTPUTS ARE TO THE μ PROCESSOR

HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PANEL ELECTRONICS
OUTPUT MUX

SIZE C | CODE IDENT NO. 27327 | DRAWING NO.

SCALE | SHEET 13 OF 16

REVISIONS

ZONE | LTR | DESCRIPTION | DATE | APPROVED

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2°
±1°

MATERIAL

FINISH–SEE NOTE

NEXT ASSY | USED ON
APPLICATION

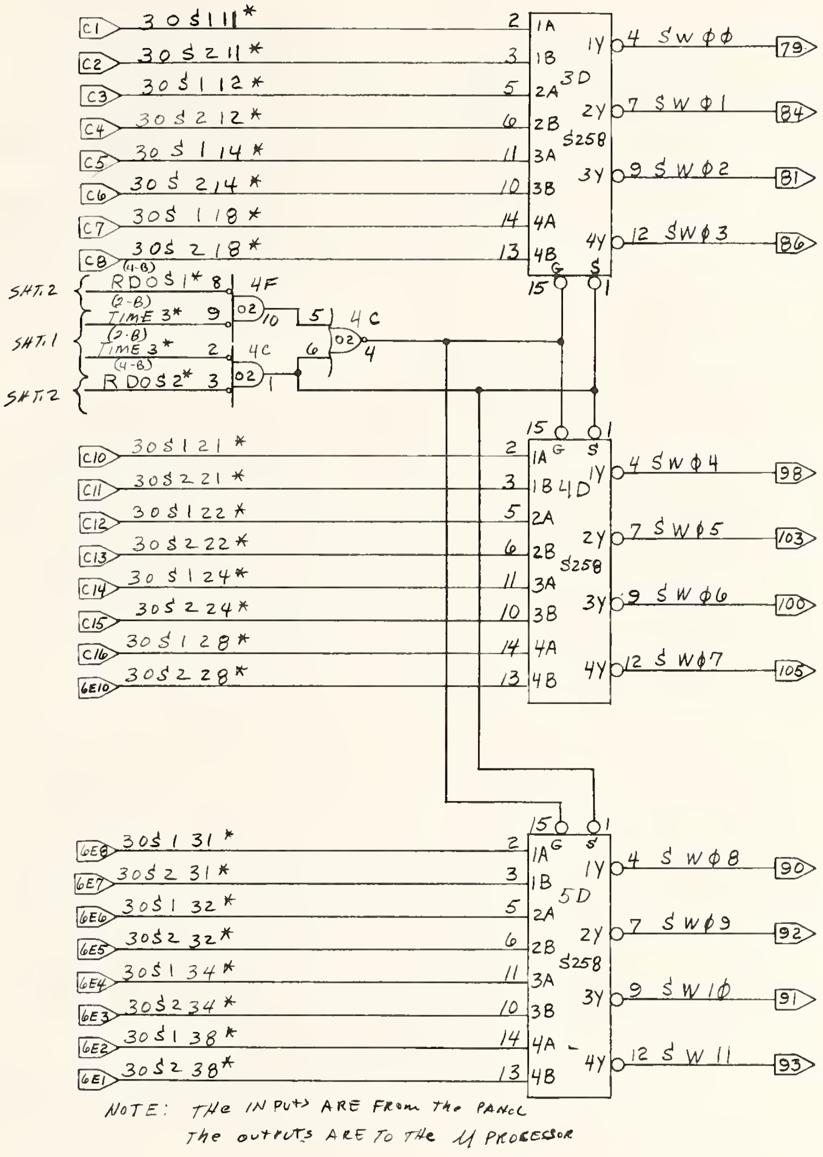
INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

HONEYWELL PART NO.

DF-8 REV. 7-75 V800

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

| ZONE | LTR | REVISIONS<br>DESCRIPTION | DATE | APPROVED |
|---|---|---|---|---|
| | | | | |



NOTE: THE INPUTS ARE FROM THE PANEL
THE OUTPUTS ARE TO THE µ PROCESSOR

HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN 55413

PANEL ELECTRONICS
OUTPUT MUX

SIZE C | CODE IDENT NO. 27327 | DRAWING NO.
SCALE | SHEET 14 OF 16

DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO

TOLERANCES UNLESS NOTED OTHERWISE
.X ± .XX ± .XXX ±
90° FORMED ANGLES ±2° ±1°
MATERIAL
FINISH–SEE NOTE

NEXT ASSY | USED ON
APPLICATION

HONEYWELL PART NO.

DF-5 REV. 7-75 V600

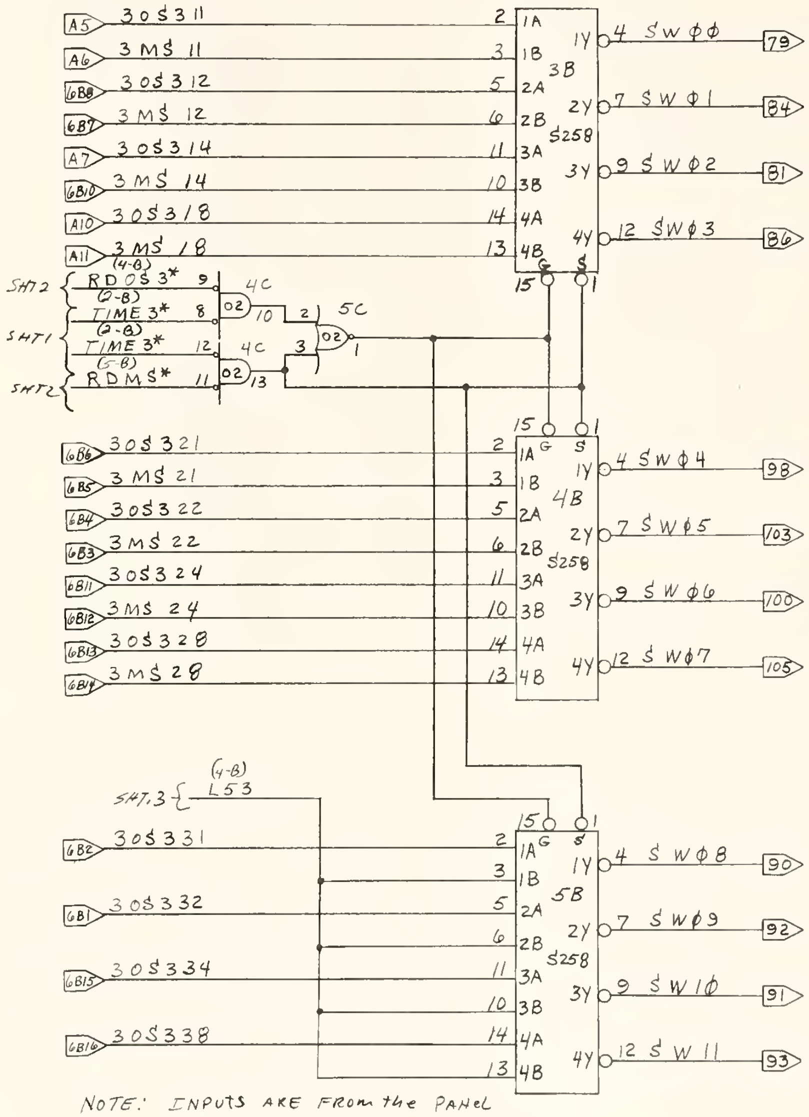
HONEYWELL INC.

SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413

PANEL ELECTRONICS
OUTPUT MUX

SIZE C · CODE IDENT NO. 27327 · DRAWING NO.

SCALE · SHEET 15 OF 16



NOTE: INPUTS ARE FROM the PANEL
OUTPUTS ARE TO The µ PROCESSOR

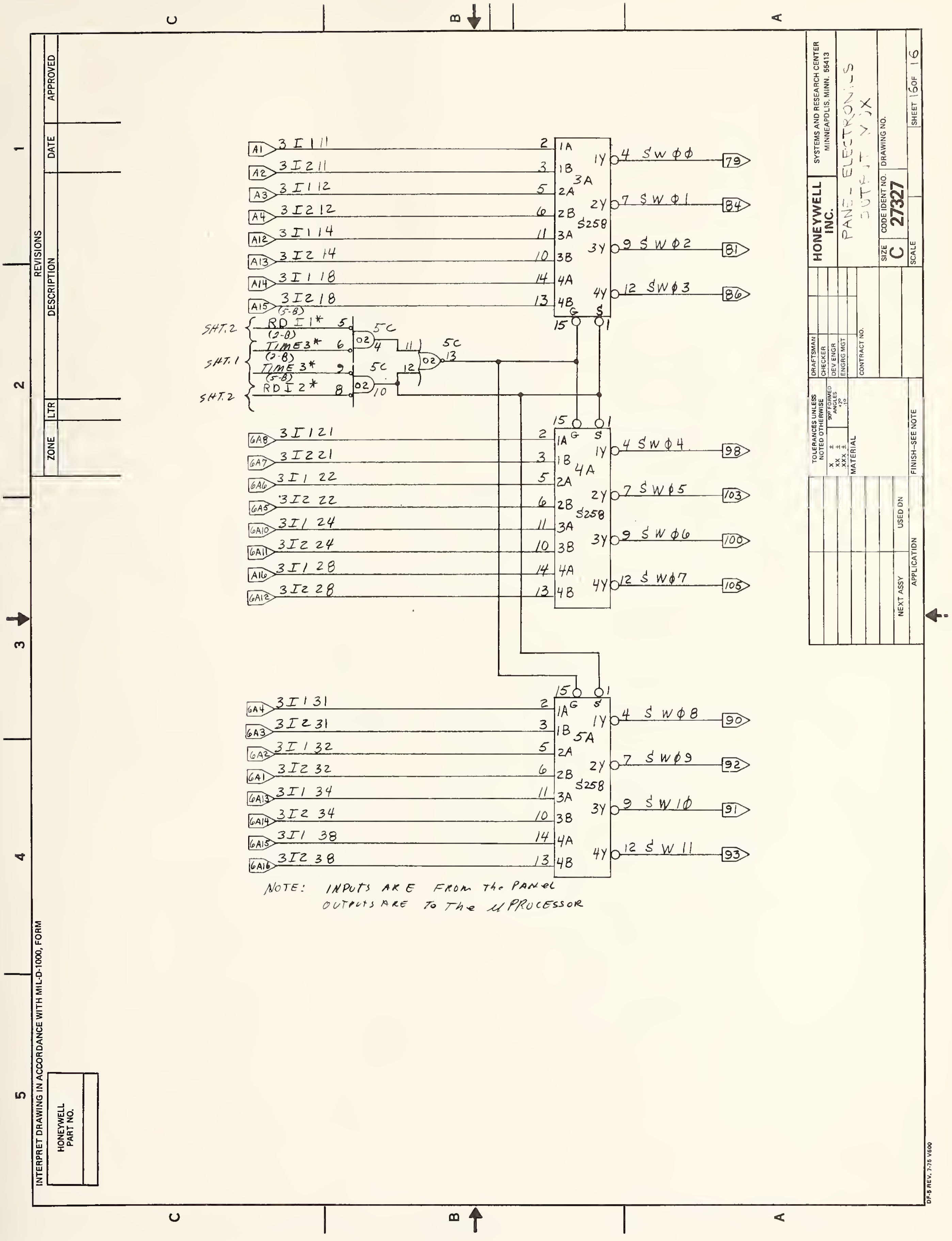
REVISIONS: ZONE · LTR · DESCRIPTION · DATE · APPROVED

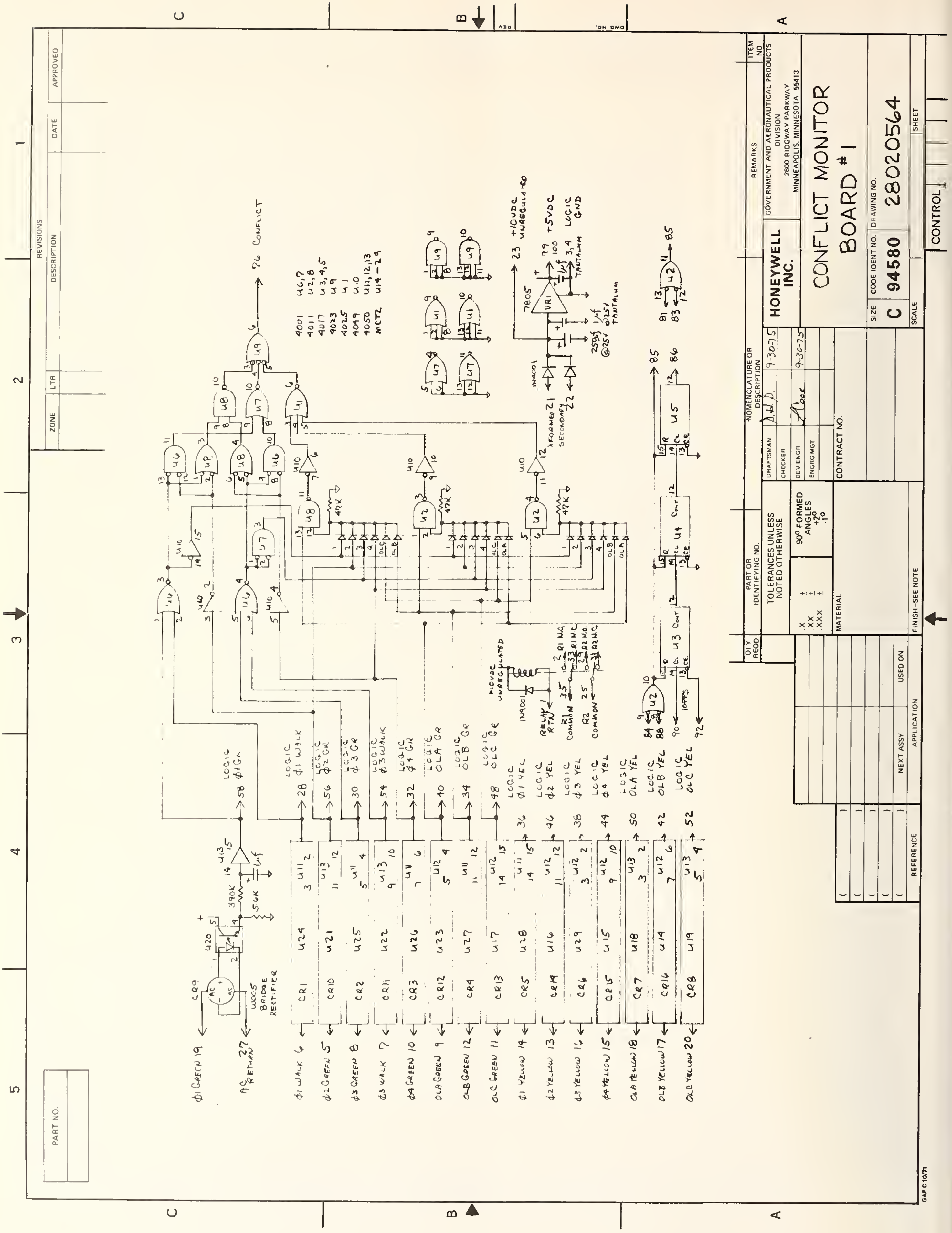
TOLERANCES UNLESS NOTED OTHERWISE · MATERIAL · FINISH–SEE NOTE · DRAFTSMAN · CHECKER · DEV ENGR · ENGRG MGT · CONTRACT NO · NEXT ASSY · USED ON · APPLICATION

HONEYWELL PART NO.

INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM

DF-8 REV. 7-78 V600


A1 3I111
A2 3I211
A3 3I112
A4 3I212
A12 3I114
A13 3I214
A14 3I118
A15 3I218
3A
S258
1Y 4 SWΦΦ 79
2Y 7 SWΦ1 84
3Y 9 SWΦ2 81
4Y 12 SWΦ3 86
SHT.2 (5-B) RD I1* 5
(2-B) TIME3* 6
SHT.1 (2-B) TIME 3* 9
SHT.2 (5-B) RD I2* 8
5C 02
5C 02 13
5C 02 10
4A
S258
6A8 3I121
6A7 3I221
6A6 3I122
6A5 3I222
6A10 3I124
6A11 3I224
A16 3I128
6A12 3I228
1Y 4 SWΦ4 98
2Y 7 SWΦ5 103
3Y 9 SWΦ6 100
4Y 12 SWΦ7 105
5A
S258
6A4 3I131
6A3 3I231
6A2 3I132
6A1 3I232
6A13 3I134
6A14 3I234
6A15 3I138
6A16 3I238
1Y 4 SWΦ8 90
2Y 7 SWΦ9 92
3Y 9 SW1Φ 91
4Y 12 SW11 93
NOTE: INPUTS ARE FROM The PANel
OUTPUTS ARE To The μPROCESSOR
REVISIONS
ZONE LTR DESCRIPTION DATE APPROVED
INTERPRET DRAWING IN ACCORDANCE WITH MIL-D-1000, FORM
HONEYWELL PART NO.
HONEYWELL INC.
SYSTEMS AND RESEARCH CENTER
MINNEAPOLIS, MINN. 55413
PANEL ELECTRONICS OUTPUT MUX
SIZE C
CODE IDENT NO. 27327
DRAWING NO.
SCALE
SHEET 16 OF 16
DRAFTSMAN
CHECKER
DEV ENGR
ENGRG MGT
CONTRACT NO.
TOLERANCES UNLESS NOTED OTHERWISE
MATERIAL
FINISH–SEE NOTE
NEXT ASSY
USED ON
APPLICATION
DF-5 REV. 7-75 V600



| QTY REQD | PART OR IDENTIFYING NO. | NOMENCLATURE OR DESCRIPTION | REMARKS | ITEM NO. |
|---|---|---|---|---|

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES ±2° ±1°
MATERIAL
FINISH-SEE NOTE

| | | |
|---|---|---|
| DRAFTSMAN | A.J.D. | 9-30-75 |
| CHECKER | | |
| DEV ENGR | Roer | 9-30-75 |
| ENGRG MGT | | |

CONTRACT NO.

HONEYWELL INC.

GOVERNMENT AND AERONAUTICAL PRODUCTS DIVISION
2600 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413

CONFLICT MONITOR BOARD #1

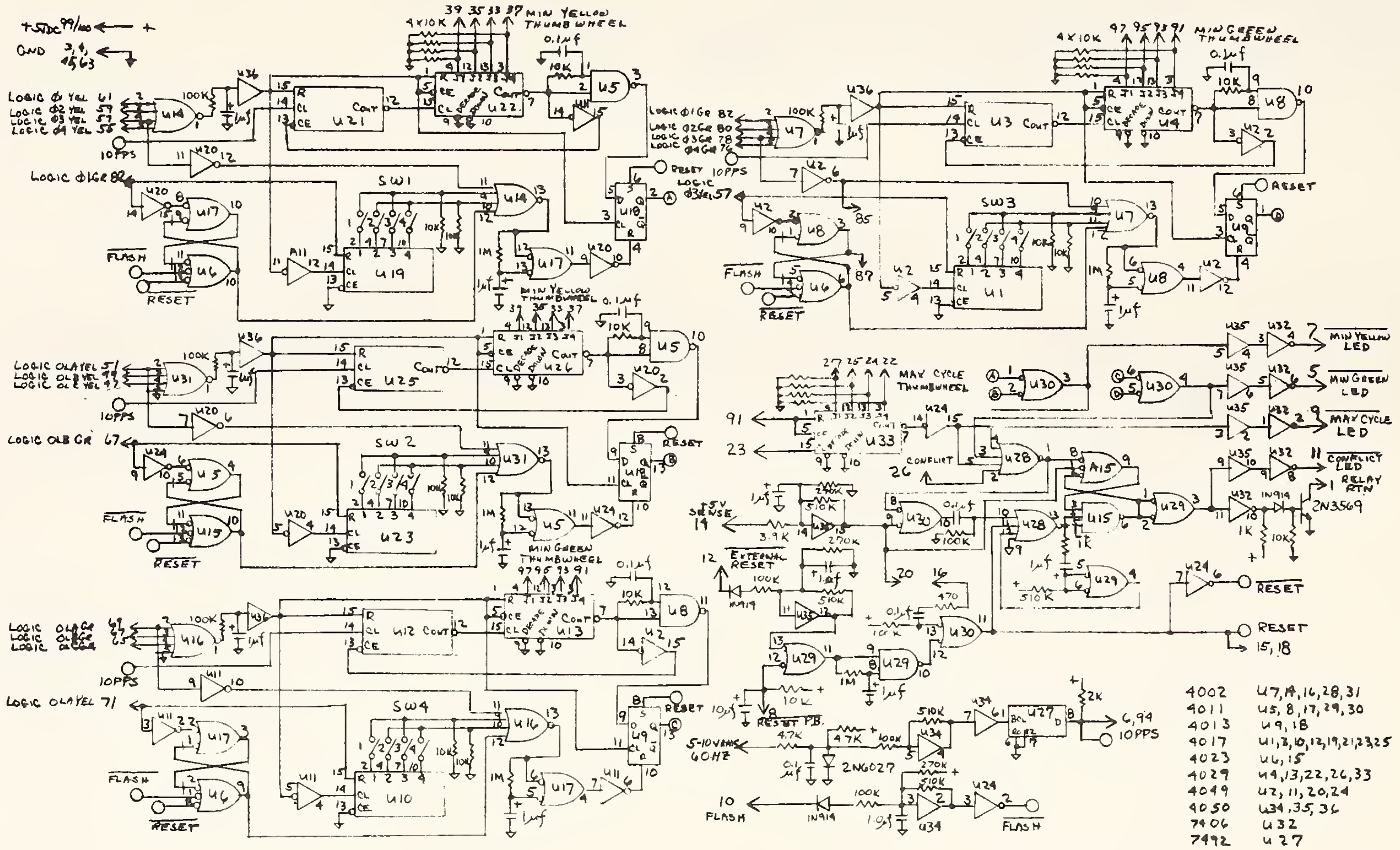
| SIZE | CODE IDENT NO. | DRAWING NO. |
|---|---|---|
| C | 94580 | 28020564 |

SCALE
SHEET
CONTROL

| NEXT ASSY | USED ON |
|---|---|
| APPLICATION | |
| REFERENCE | |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |

PART NO.



| | |
|---|---|
| 4002 | U7,14,16,28,31 |
| 4011 | U5, 8, 17, 29, 30 |
| 4013 | U9, 18 |
| 4017 | U1,3,10,12,19,21,23,25 |
| 4023 | U6, 15 |
| 4029 | U4,13,22,26,33 |
| 4049 | U2, 11, 20, 24 |
| 4050 | U34, 35, 36 |
| 7406 | U32 |
| 7492 | U27 |

| QTY REQD | PART OR IDENTIFYING NO | NOMENCLATURE OR DESCRIPTION | ITEM NO. |
|---|---|---|---|

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES +2° -1°

MATERIAL

FINISH—SEE NOTE

| | |
|---|---|
| DRAFTSMAN | D.H.D. 9-30-75 |
| CHECKER | |
| DEV ENGR | J. Cook 9-30-75 |
| ENGRG MGT | |

CONTRACT NO.

NEXT ASSY | USED ON

APPLICATION

REFERENCE

**HONEYWELL INC.**

GOVERNMENT AND AERONAUTICAL PRODUCTS DIVISION
2600 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413

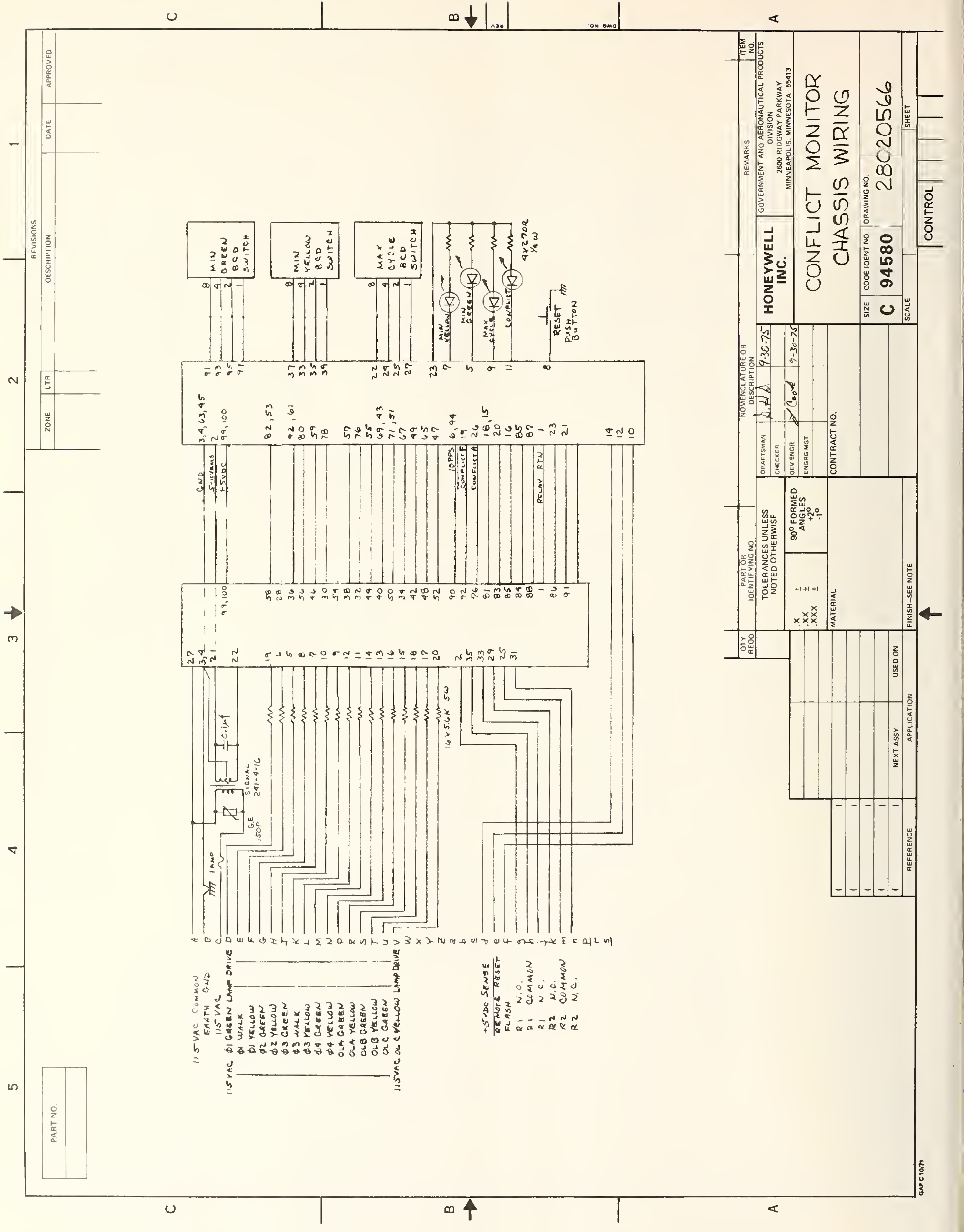
# CONFLICT MONITOR BOARD #2

| SIZE | CODE IDENT NO | DRAWING NO. |
|---|---|---|
| C | 94580 | 28020565 |

SCALE | SHEET

CONTROL

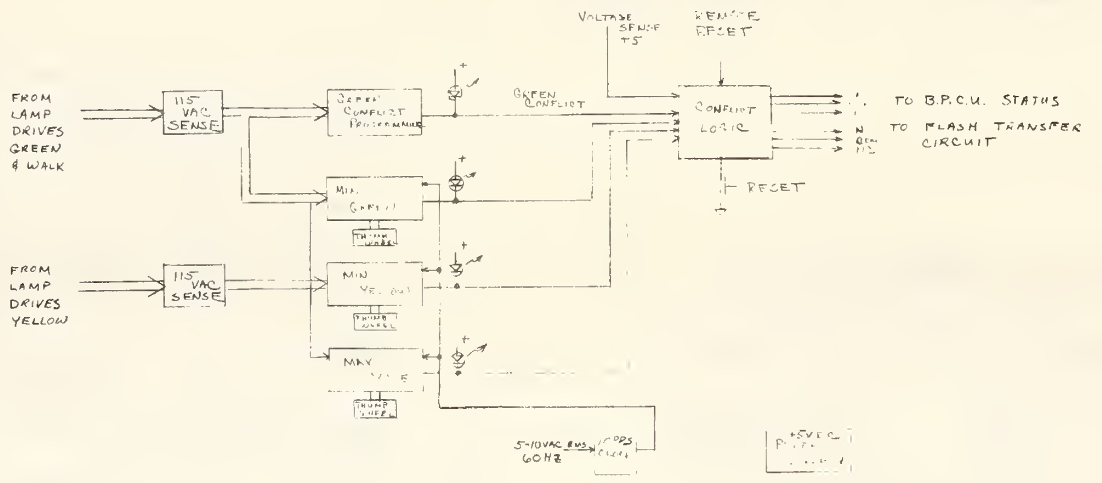
GAP C 10/71



CONFLICT MONITOR
CHASSIS WIRING
HONEYWELL INC.
GOVERNMENT AND AERONAUTICAL PRODUCTS DIVISION
2600 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413
SIZE C
CODE IDENT NO 94580
DRAWING NO. 28020566
SCALE
SHEET
CONTROL
DRAFTSMAN 9-30-75
CHECKER
DEV ENGR 9-30-75
ENGRG MGT
CONTRACT NO.
TOLERANCES UNLESS NOTED OTHERWISE
90° FORMED ANGLES +2° -1°
MATERIAL
FINISH–SEE NOTE
NEXT ASSY
USED ON
APPLICATION
REFERENCE
QTY REQD
PART OR IDENTIFYING NO
NOMENCLATURE OR DESCRIPTION
REMARKS
ITEM NO.
REVISIONS
ZONE
LTR
DESCRIPTION
DATE
APPROVED
PART NO.
REV
DWG NO.
MIN GREEN BCD SWITCH
MIN YELLOW BCD SWITCH
MAX CYCLE BCD SWITCH
4x270Ω 1/4 W
MIN YELLOW
MIN GREEN
MAX CYCLE
CONFLICT
RESET PUSH BUTTON
GND
5-10VRMS
+5VDC
10PPS
CONFLICT F
CONFLICT A
RELAY RTN
C 0.1μf
SIGNAL 241-4-16
G.E. 150P
1 AMP
16x5.6K 5W
115VAC COMMON
EARTH GND
115VAC
115VAC Ø1 GREEN LAMP DRIVE
Ø1 WALK
Ø1 YELLOW
Ø2 GREEN
Ø2 YELLOW
Ø3 GREEN
Ø3 WALK
Ø3 YELLOW
Ø4 GREEN
Ø4 YELLOW
OLA GREEN
OLA YELLOW
OLB GREEN
OLB YELLOW
OLC GREEN
115VAC OLC YELLOW LAMP DRIVE
+5VDC SENSE
REMOTE RESET
FLASH
R1 N.O.
R1 COMMON
R1 N.C.
R2 N.O.
R2 COMMON
R2 N.C.
GAP C 10/71

| PART NO. |
|---|
| |

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |
| | | | | |



| QTY REQD | PART OR IDENTIFYING NO | NOMENCLATURE OR DESCRIPTION | REMARKS | ITEM NO |
|---|---|---|---|---|
| | | | | |

| | | |
|---|---|---|
| DRAFTSMAN | D.H.D. | 9-30-75 |
| CHECKER | | |
| DEV ENGR | L Cook | 9-30-75 |
| ENGRG MGT | | |
| CONTRACT NO. | | |

TOLERANCES UNLESS NOTED OTHERWISE

X ±
XX ±
.XXX ±

90° FORMED ANGLES ±2° .1°

MATERIAL

FINISH SEE NOTE

| REFERENCE | NEXT ASSY | USED ON |
|---|---|---|
| ( ) | | |
| ( ) | | |
| ( ) | | |
| ( ) | | |
| ( ) | | |
| | APPLICATION | |

**HONEYWELL INC.**

GOVERNMENT AND AERONAUTICAL PRODUCTS DIVISION
2600 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413

# CONFLICT MONITOR BLOCK DIAGRAM

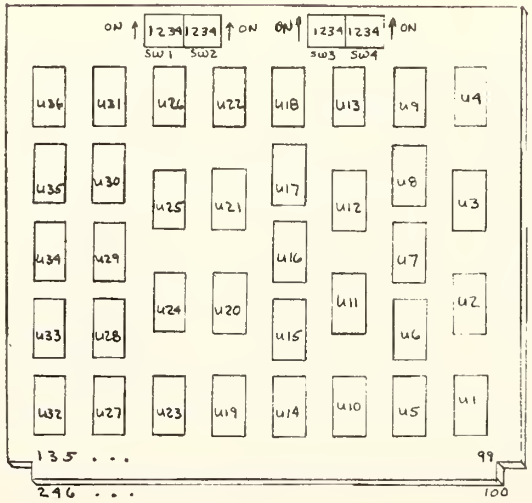
| SIZE | CODE IDENT NO | DRAWING NO. |
|---|---|---|
| C | 94580 | 28020567 |
| SCALE | | SHEET |

CONTROL

GAP C 10/71

| REVISIONS | | | | |
|---|---|---|---|---|
| ZONE | LTR | DESCRIPTION | DATE | APPROVED |

PART NO.



BD #2

SWITCHES 1 thru 4 ARE USED TO DETERMINE IN PART A VIOLATION OF MINIMUM GREENS OR YELLOWS.

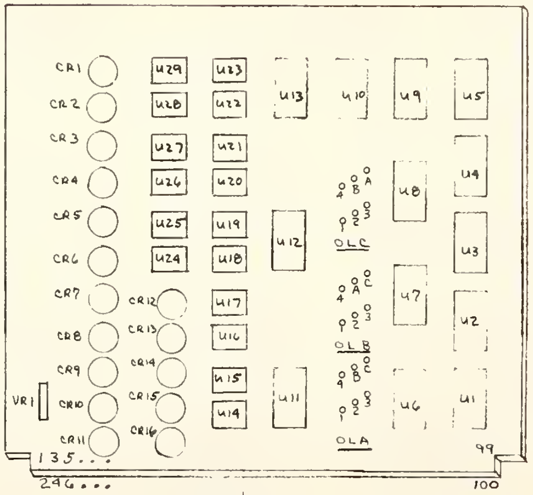
1) SET SW1 TO THE NUMBER OF YELLOWS PER CYCLE FROM PHASES 1 thru 4.

2) SET SW2 TO THE NUMBER OF YELLOWS PER CYCLE FROM OVERLAP PHASES A thru C.

3) SET SW3 TO THE NUMBER OF GREENS PER CYCLE FROM PHASES 1 thru 4.

4) SET SW4 TO THE NUMBER OF GREENS PER CYCLE FROM OVERLAP PHASES A thru C.

IF THE NUMBER OF GREENS OR YELLOWS VARIES WITH DIFFERENT CYCLES TWO SWITCHES MAY BE ON IN ANY ONE SWITCH SET. THE SWITCHES THAT ARE ON MUST HOWEVER BE ADJACENT.



BD #1

WHEN OVERLAPS ARE USED THE DIODES WHICH CORRESPOND TO THE NON-CONFLICTING PHASES MUST BE REMOVED.
Ex. IF OLA GREEN IS ON WHEN ϕ1 GREEN OR ϕ2 GREEN IS ON THEN THE DIODE LABELED 1 AND THE DIODE LABELED 2 MUST BE REMOVED FROM THE OLA SECTION.

| QTY REQD | PART OR IDENTIFYING NO | NOMENCLATURE OR DESCRIPTION | REMARKS | ITEM NO |
|---|---|---|---|---|

TOLERANCES UNLESS NOTED OTHERWISE
.X ±
.XX ±
.XXX ±
90° FORMED ANGLES +2° -1°
MATERIAL
FINISH SEE NOTE

| | | |
|---|---|---|
| DRAFTSMAN | D.H.D. | 9-30-75 |
| CHECKER | | |
| DEV ENGR | L Cook | 9-30-75 |
| ENGRG MGT | | |

CONTRACT NO.

HONEYWELL INC.
GOVERNMENT AND AERONAUTICAL PRODUCTS DIVISION
2600 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413

CONFLICT MONITOR P.C. CARD ASSY (BD #1 & #2)

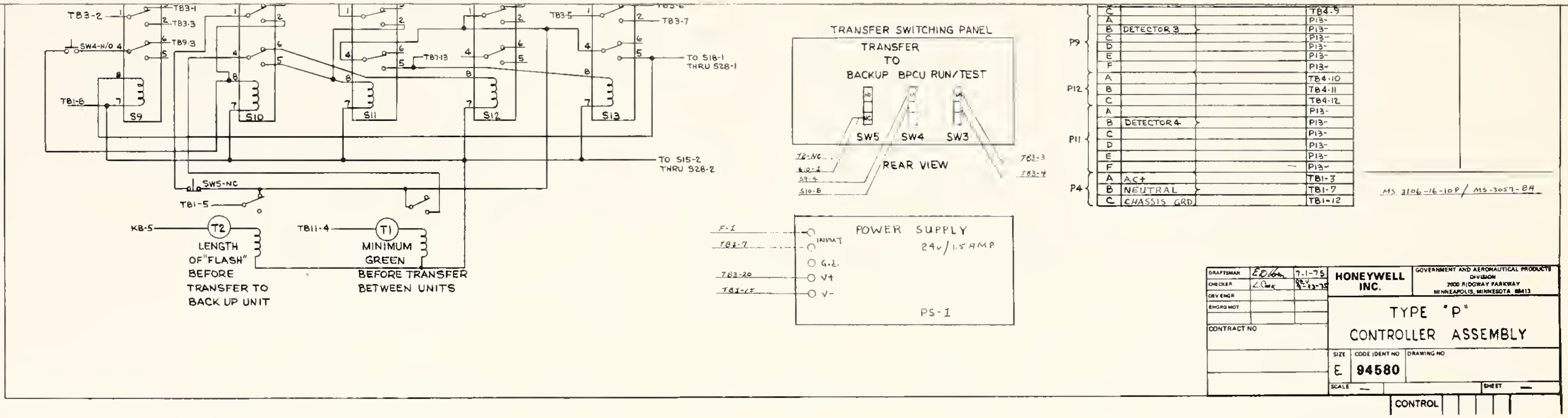
| SIZE | CODE IDENT NO | DRAWING NO. |
|---|---|---|
| C | 94580 | 28020568 |

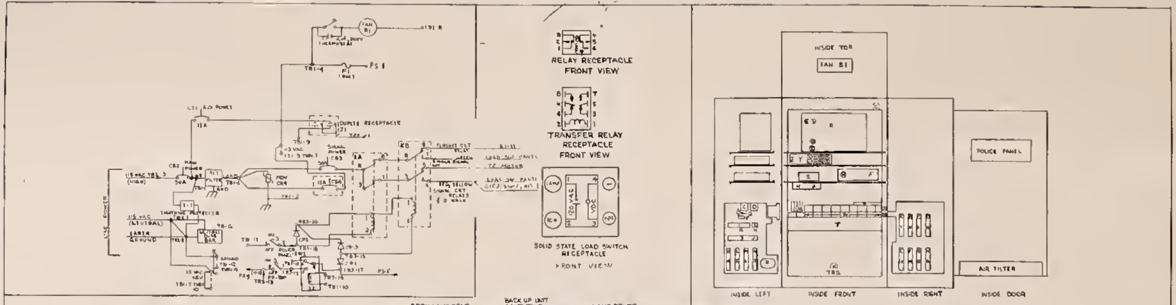
SCALE
SHEET

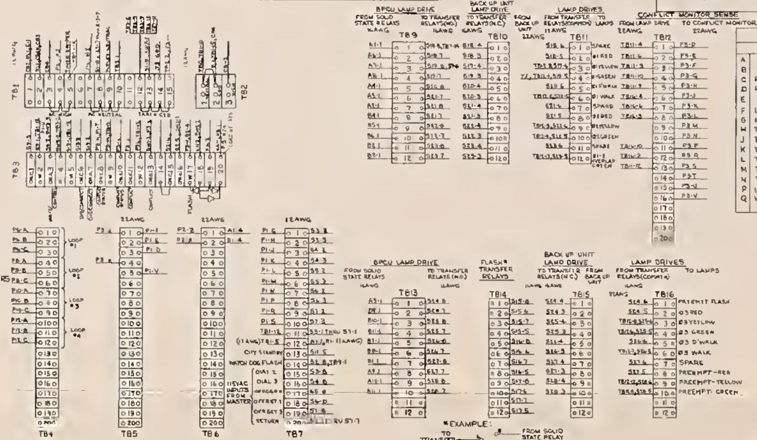
NEXT ASSY
USED ON
APPLICATION
REFERENCE

CONTROL

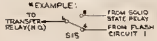
GAP C 10/71


TB3-2
TB3-1
TB3-3
TB9-3
SW4-N/O
TB1-6
S9
S10
S11
S12
S13
TB7-13
TB3-5
TB3-7
TO S18-1
THRU S28-1
TO S15-2
THRU S28-2
SW5-NC
TB1-5
K8-5
T2
LENGTH
OF "FLASH"
BEFORE
TRANSFER TO
BACK UP UNIT
TB11-4
T1
MINIMUM
GREEN
BEFORE TRANSFER
BETWEEN UNITS
TRANSFER SWITCHING PANEL
TRANSFER
TO
BACKUP BPCU RUN/TEST
NO
NC
SW5
SW4
SW3
REAR VIEW
T2-NC
S10-1
S9-4
S10-8
TB3-3
TB3-4
F-1
TB1-7
TB3-20
TB1-15
POWER SUPPLY
24v/1.5AMP
INPUT
G.L.
V+
V-
PS-1
P9
C TB4.9
A P13-
B DETECTOR 3 P13-
C P13-
D P13-
E P13-
F P13-
P12
A TB4-10
B TB4-11
C TB4-12
P11
A P13-
B DETECTOR 4 P13-
C P13-
D P13-
E P13-
F P13-
P4
A AC+ TB1-3
B NEUTRAL TB1-7
C CHASSIS GRD TB1-12
MS 3106-16-10P / MS-3057-8A
DRAFTSMAN E D 7-1-75
CHECKER REV 9-23-75
DEV ENGR
ENGRG MGT
CONTRACT NO
HONEYWELL INC.
GOVERNMENT AND AERONAUTICAL PRODUCTS DIVISION
2600 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413
TYPE "P"
CONTROLLER ASSEMBLY
SIZE E
CODE IDENT NO 94580
DRAWING NO
SCALE —
SHEET —
CONTROL





*EXAMPLE:



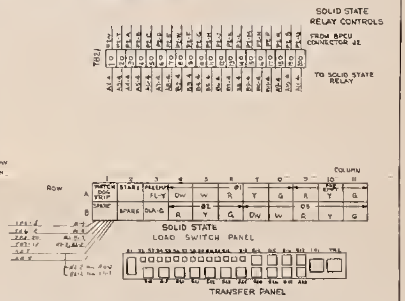
TO CAUSE Ø1 YELLOW TO FLASH DURING FLASHING INTERVALS -

1)REMOVE WIRE ON TB9-3 COMING FROM S19-B.

2)CONNECT WIRE FROM S19-8 TO A COMMON POSITION(PIN 5 OR 6) OF S15,16, OR 17.

3)ADD A WIRE FROM THE N.O. POSITION(PIN 7 OR 8) OF S15,16,OR 17 TO TB9-3.

| LEGEND | | | | | | | |
|---|---|---|---|---|---|---|---|
| INSIDE LEFT BOX | | INSIDE FRONT | | | | INSIDE RIGHT SIDE | |
| A | | A | 115 AC DUPLEX | | | A | TERMINAL BOARD, TB9 |
| B | | B | CONTROLLER UNIT | | | B | TERMINAL BOARD, TB10 |
| C | RELAY, KA | C | SWITCH – THERMOSTAT | | | C | TERMINAL BOARD, TB11 |
| D | RELAY, KB | D | CONNECTOR, P1 | T | TRANSFER PANEL | D | TERMINAL BOARD, TB12 |
| E | | E | CONNECTOR, P2 | U | CIRCUIT BREAKER PANEL | E | TERMINAL BOARD, TB13 |
| F | LIGHTNING ARRESTOR E1 | F | CONFLICT MONITOR | V | TERMINAL BOARD, TB21 | F | TERMINAL BOARD, TB14 |
| G | TERMINAL BOARD, TB1 | G | LOOP DETECTOR | W | LOAD SWITCH PANEL ASSY | G | TERMINAL BOARD, TB15 |
| H | TERMINAL BOARD, TB2 | H | CONNECTOR, P3 | X | NEUTRAL BUSS BAR, TB8 | H | TERMINAL BOARD, TB16 |
| J | TERMINAL BOARD, TB3 | J | CONNECTOR, P4 | a | CONNECTOR, P13 | | |
| K | TERMINAL BOARD, TB4 | K | CONNECTOR, P5 | b | CONNECTOR, P14 | | |
| L | TERMINAL BOARD, TB5 | L | CONNECTOR, P6 | Y | LOOP AMPLIFIER | | |
| M | TERMINAL BOARD, TB6 | M | CONNECTOR, P7 | Z | TRANSFER SW | | |
| N | TERMINAL BOARD, TB7 | N | CONNECTOR, P8 | ε | POWER SUPPLY | | |
| P | LINE FILTER, FL1 | P | CONNECTOR, P9 | | | | |
| Q | VARACTOR, CR4 | Q | CONNECTOR, P10 | | | | |
| | | R | CONNECTOR, P11 | | | | |
| | | S | CONNECTOR, P12 | | | | |



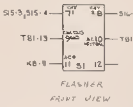
| CONTROLLER UNIT CONNECTOR, P1 | | | 37 PINS | MS3106B 18-21S | |
|---|---|---|---|---|---|
| PIN | FUNCTION | TERMIN. | PIN | FUNCTION | TERMIN. |
| A | AC COMMON | TB1-9 | d | | |
| B | CHASSIS GND | TB1-14 | e | | |
| C | AC+ | TB1-4 | f | | |
| D | SPARE | TB9-5 | g | | |
| E | LOGIC GND | TB5-2 | h | | |
| F | +5VDC | TB5-1 | j | | |
| G | DIAL 2 N.O. | TB7-1 | k | | |
| H | DIAL 2 N.C. | TB7-2 | m | | |
| J | DIAL 3 N.O. | TB7-3 | n | | |
| K | DIAL 3 N.C. | TB7-4 | p | | |
| L | OFFSET 1 N.O. | TB7-5 | r | | |
| M | OFFSET 1 N.C. | TB7-6 | s | | |
| N | OFFSET 2 N.O. | TB7-7 | | | |
| P | OFFSET 2 N.C. | TB7-8 | | | |
| R | OFFSET 3 N.O. | TB7-9 | | | |
| S | OFFSET 3 N.C. | TB7-10 | | | |
| T | CONFLICT | TB9-8 | | | |
| U | WAIT/CONTROL | TB5-3 | | | |
| V | CONFLICT RESET | TB5-5 | | | |
| W | | | | | |
| X | | | | | |
| Z | | | | | |
| a | | | | | |
| b | | | | | |
| c | | | | | |

TWISTED

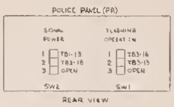
| CONTROLLER UNIT CONNECTOR, P2 | | | 37 PINS | MS3106D-28-21SY | |
|---|---|---|---|---|---|
| PIN | FUNCTION | TERMIN. | PIN | FUNCTION | TERMIN. |
| A | OUTPUT 1 Ø1 DON'T WALK | TB10-3 | d | | |
| B | 2 WALK | -4 | e | | |
| C | 3 RED | -5 | f | | |
| D | 4 YELLOW | -6 | g | | |
| E | 5 GREEN | -7 | h | | |
| F | 6 Ø1A GREEN | -9 | j | | |
| G | 7 Ø2 RED | -10 | k | | |
| H | 8 YELLOW | -11 | m | | |
| J | 9 GREEN | -12 | n | | |
| K | 10 Ø3 DON'T WALK | -13 | p | | |
| L | 11 WALK | -14 | r | | |
| M | 12 RED | -15 | s | +24V RETURN | TB1-15 |
| N | 13 YELLOW | -16 | | | |
| P | 14 GREEN | -17 | | | |
| R | 15 PREEMPT RED | -18 | | | |
| S | 16 YELLOW | -19 | | | |
| T | 17 [illegible] | -20 | | | |
| U | 18 GREEN | -20 | | | |
| V | 19 SPARE | -1 | | | |
| W | 20 SPARE | -8 | | | |
| X | SPARE | TB6-2 | | | |
| Z | WATCH DOG TRIP | TB6-1 | | | |
| a | | | | | |
| b | | | | | |
| c | | | | | |

TWISTED

| CONFLICT MONITOR P3 | | | 37 PINS | MS3106B-28-21S | |
|---|---|---|---|---|---|
| PIN | FUNCTION | TERMINAL | PIN | FUNCTION | TERMIN. |
| A | AC COMMON | TB1-9 | d | +5VDC STAGE | TB5-1 |
| B | CHASSIS GND | TB1-14 | e | REMOTE RESET | TB5-4 |
| C | AC+ | TB1-4 | f | FLASH | TB5-7 |
| D | Ø1 GREEN | TB12-1 | g | R1 N.O. | TB5-13 |
| E | Ø1 WALK | -2 | h | R1 COMMON | TB5-12 |
| F | Ø1 YELLOW | -3 | j | R1 N.C. | TB5-11 |
| G | Ø2 GREEN | -4 | k | R2 N.O. | TB5-10 |
| H | Ø2 YELLOW | -5 | m | R2 COMMON | TB5-9 |
| J | Ø3 GREEN | -6 | n | R2 N.C. | TB5-8 |
| K | Ø3 WALK | -7 | p | | |
| L | Ø3 YELLOW | -8 | r | | |
| M | Ø4 GREEN | -9 | s | | |
| N | Ø4 YELLOW | -10 | | | |
| P | Ø1A GREEN | -11 | | | |
| R | Ø1A YELLOW | -12 | | | |
| S | Ø1B GREEN | -13 | | | |
| T | Ø1B YELLOW | -14 | | | |
| U | Ø1C GREEN | -15 | | | |
| V | Ø1C YELLOW | -16 | | | |
| W | | | | | |
| X | | | | | |
| Z | | | | | |
| a | | | | | |
| b | | | | | |
| c | | | | | |

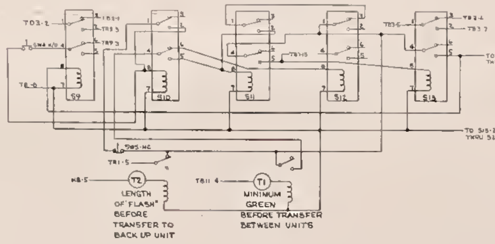


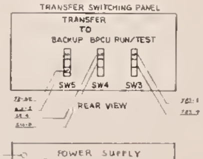
NOTE
CIRCUITS 1&2 ARE THE FLASHER OUTPUTS. EACH CIRCUIT SUPPLIES 10AMPS MAX. THE TWO CIRCUITS FLASH AT A ONE PULSE PER SECOND RATE THERE IS A 180° PHASE SHIFT BETWEEN THE TWO CIRCUITS.

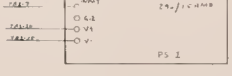


| | INTERSECTION DETECTOR | | |
|---|---|---|---|
| | PIN | FUNCTION | TERMIN. |
| P6 | A | | TB4-1 |
| | B | | TB4-2 |
| | C | | TB4-3 |
| P5 | A | DETECTOR 1 | P13- |
| | B | | P13- |
| | C | | P13- |
| | D | | P13- |
| | E | | P13- |
| | F | | P13- |
| P8 | A | | TB4-4 |
| | B | | TB4-5 |
| | C | | TB4-6 |
| P7 | A | | P13- |
| | B | DETECTOR 2 | P13- |
| | C | | P13- |
| | D | | P13- |
| | E | | P13- |
| | F | | P13- |
| P10 | A | | TB4-7 |
| | B | | TB4-8 |
| | C | | TB4-9 |
| P9 | A | | P13- |
| | B | DETECTOR 3 | P13- |
| | C | | P13- |
| | D | | P13- |
| | E | | P13- |
| | F | | P13- |
| P12 | A | | TB4-10 |
| | B | | TB4-11 |
| | C | | TB4-12 |
| P11 | A | | P13- |
| | B | DETECTOR 4 | P13- |
| | C | | P13- |
| | D | | P13- |
| | E | | P13- |
| | F | | P13- |
| P4 | A | AC+ | TB1-3 |
| | B | NEUTRAL | TB1-7 |
| | C | CHASSIS GND | TB1-12 |

MS 3106 16-10S / MS 3117-8W







| | | | HONEYWELL INC | |
|---|---|---|---|---|
| | | 9-1-75 | | |
| CONTRACT NO | | | TYPE "P" CONTROLLER ASSEMBLY | |
| | | | E | 94580 |
| | | | CONTROL | |

# APPENDIX IV

Order Number IMP-16C/921C
Publication Number 4200021C

Integrated MicroProcessor-16C

# IMP-16C
# APPLICATION MANUAL

January 1974

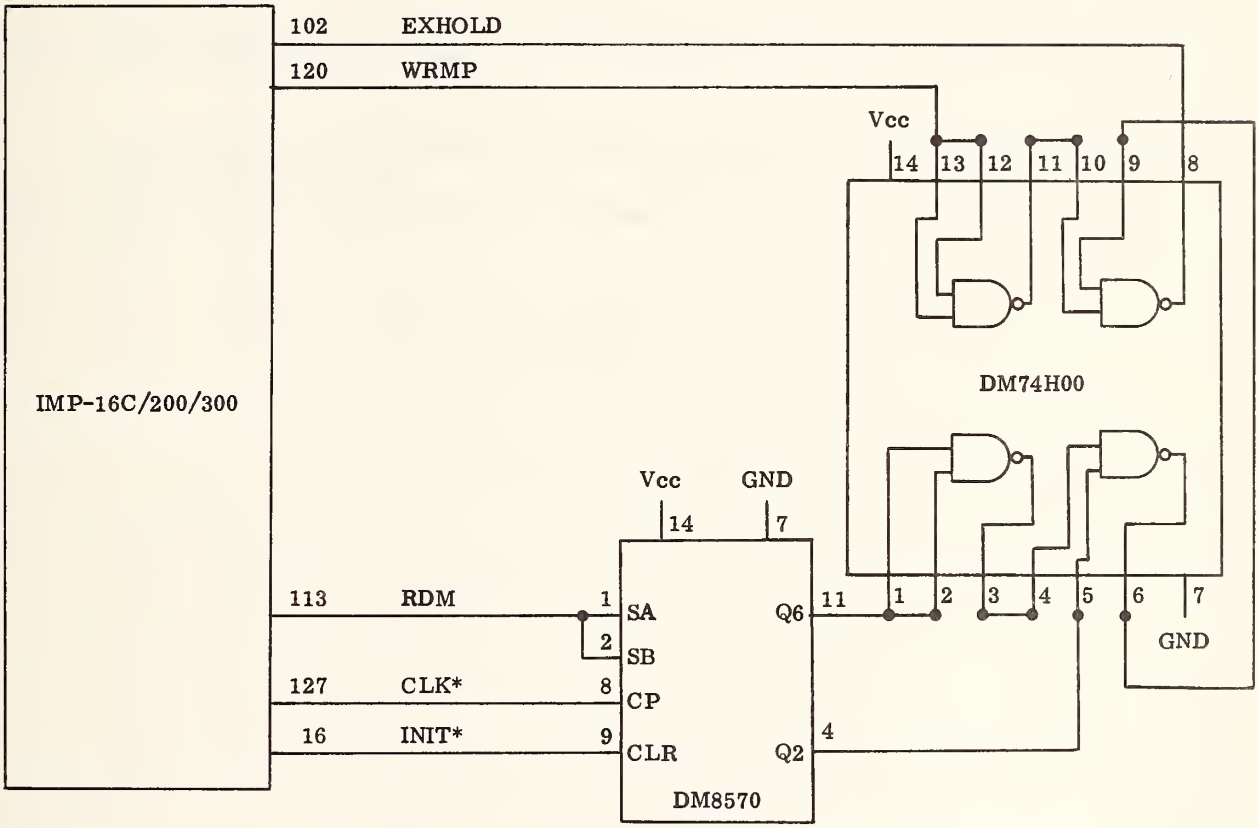
©National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, California 95051

September 1974

## NOTICE

IMP-16C Application Manual
(Pub. No. 4200021C)

Refer to paragraph 9.5. Note that the ROMs or PROMs used with the IMP-16C Microprocessor require an additional clock delay if the access time is greater than 850 nanoseconds. The circuit to extend the clock period for ROMs or PROMs that require access times up to 1.1 microseconds is shown below. It is required to remove Jumper W6 on the IMP-16C P.C. card when implementing this circuit.



Circuit To Extend Clock Period

# PREFACE

The IMP-16C Application Manual provides information required for a user to become familiar with the IMP-16C microprocessor functional and logic circuits, instruction set, general input/output interfacing, and a general system verification. With this information, the user may adapt the IMP-16C microprocessor to his particular application(s).

This issue, order number IMP-16C/921C, pertains to the IMP-16C/200 and IMP-16/300 microprocessors. The IMP-16C/200 is a pin-compatible version of the earlier IMP-16C card, with the layout changed to accommodate a second Control Read Only Memory (CROM). This second CROM may be supplied with an extended instruction set, or the second CROM can be customized for special applications. The IMP-16C/200 is supplied with only one CROM (programmed for the basic instruction set) and with an empty CROM socket. The IMP-16C/300 has two CROMs, the second CROM programmed for the extended instruction set and inserted in the second CROM socket. This is the only difference between the two cards.

The IMP-16C Interfacing Guide (formerly issued as publication number 4200035A) is presented as Supplement 1 to this manual.

The material in this manual is for information purposes only and is subject to change without notice.

Copies of this publication and other National Semiconductor publications may be obtained from the sales offices listed on the back cover.

## CONTENTS

Chapter Page

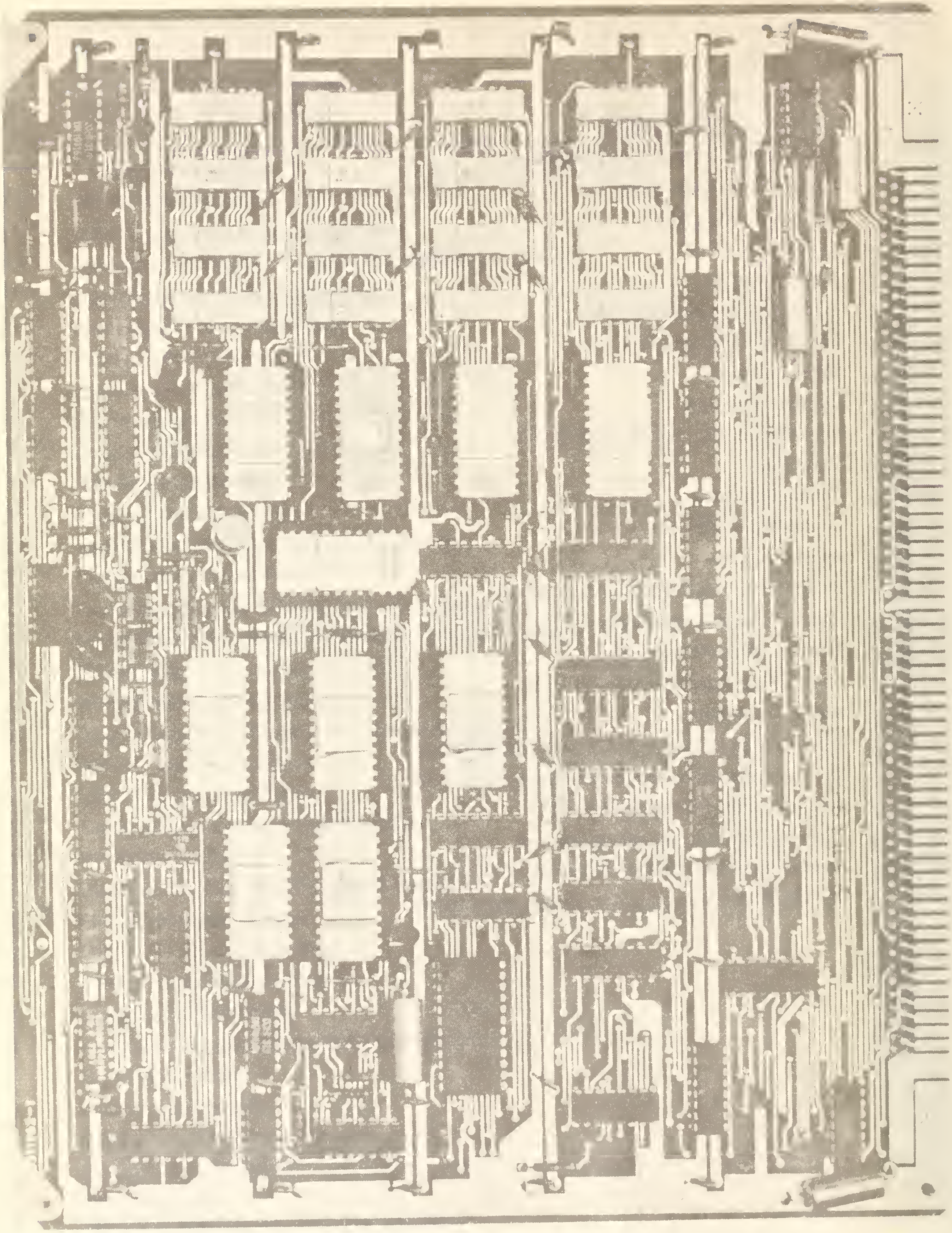
# CONTENTS (Continued)

## CONTENTS (Continued)

Chapter Page

# ILLUSTRATIONS

# TABLES



NS00145

*IMP-16C Card*

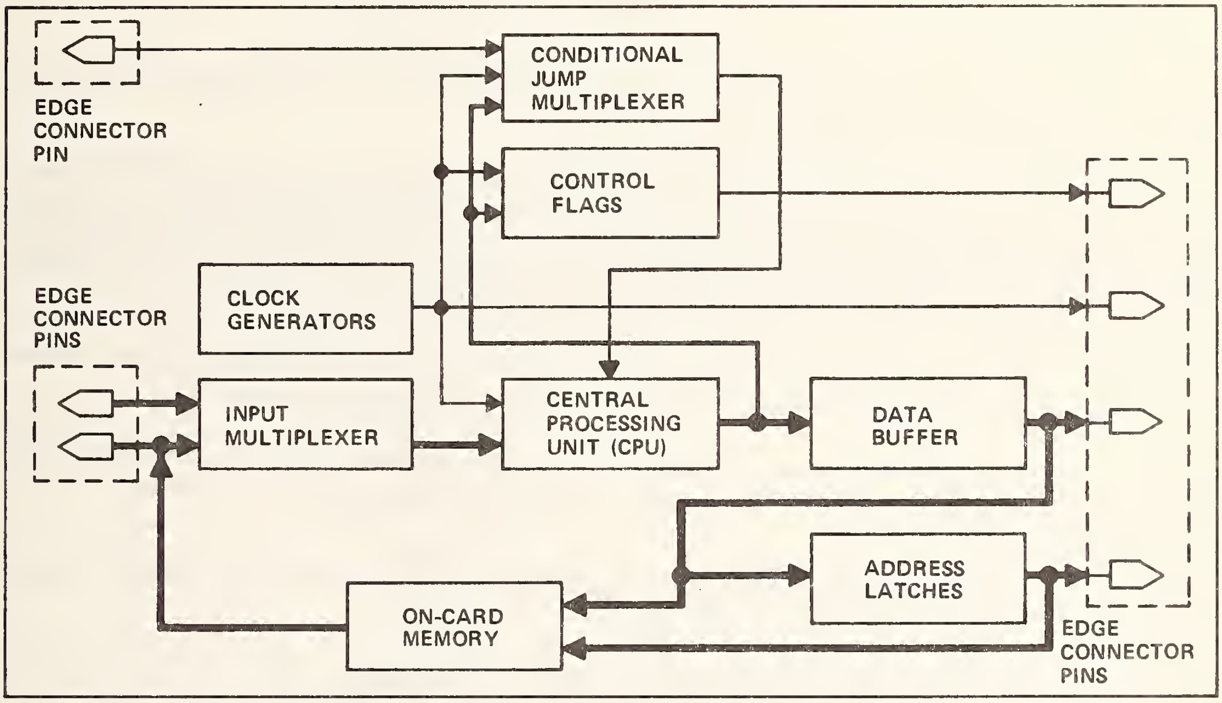
# Chapter 1
# GENERAL INFORMATION

## 1.1 IMP-16C CONFIGURATION

The IMP-16C is a 16-bit parallel processor. It is packaged on an 8½-by-11-inch printed wiring card, which is shown on the facing page. A 144-pin connector is located on the edge of the card for connecting the IMP-16C circuits to interfacing units.

The major functional units of the IMP-16C are shown in figure 1–1 and are composed of the following:

- Central Processing Unit (CPU)
- Clock Generators
- Input Multiplexer
- Data Buffer
- Control Flags
- Conditional Jump Multiplexer
- On-Card Memory
- Address Latches



NS00121

*Figure 1–1. IMP-16C Major Functional Units*

The CPU is configured around the National Semiconductor GPC/P (General-Purpose Controller/Processor) MOS/LSI devices, as shown in the simplified block diagram of figure 1–2. The MOS/LSI devices consist of one CROM (Control Read Only Memory) and four RALUs (Register and Arithmetic Logic Units). Each RALU handles 4 bits, and a 16-bit unit is formed by connecting four RALUs in parallel. A 4-bit-wide control bus is used by the CROM to communicate most of the control information to the RALUs. The CPU includes provision for adding a second CROM for an optional extended instruction set.

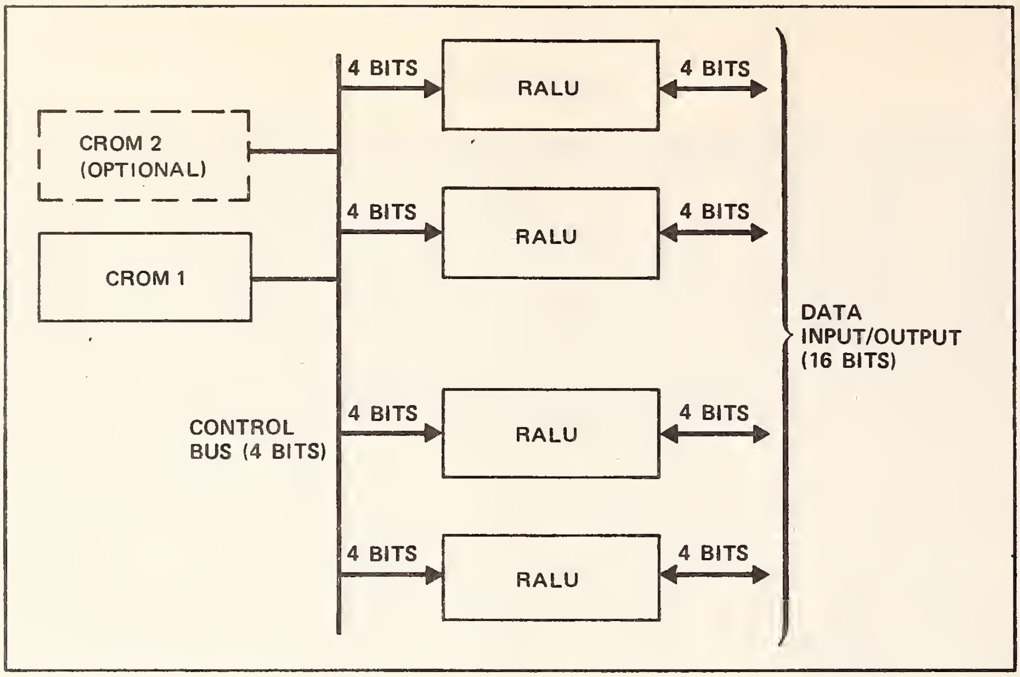


*Figure 1–2. IMP-16C CPU Components*

The Clock Generator provides the MOS clock drivers and CPU timing signals. The system clock is distributed outside of the IMP-16C for synchronization of peripheral units with the IMP-16C.

External to the MOS/LSI circuits but still within the IMP-16C are control flags for both the IMP-16C and external interfacing circuits. These control flags are in addition to the status flags that are internal to the RALUs. The status flags are discussed in the description of the CPU in chapter 2. The control flags are discussed in chapter 3 under the control flag instructions. Conditional branches are selected by the Conditional Jump Multiplexer. These branch conditions are discussed in chapter 3 under the Branch-On-Condition (BOC) instruction.

Data from the user's peripheral devices and add-on memory are received by the Input Multiplexer. Data from the On-Card Memory are also processed through the Input Multiplexer en route to the Central Processing Unit.

Output data are made available from the 16-bit Data Buffer via the card-edge connector to the user's peripheral devices and add-on memory. A 16-bit address bus is also brought out to the card-edge connector for addressing both add-on memory and peripheral devices.

The memory on the IMP-16C card consists of 256 words of read/write memory and sockets for 512 words of PROM or ROM. A maximum of 65,536 words may be addressed.

Chapter 2 contains a more-detailed functional description of the IMP-16C. The instruction set is explained in chapter 3. Chapter 4 presents a circuit-level description.

## 1.2 IMP-16C OPERATIONAL FEATURES

| | |
|---|---|
| *Word Length* | 16 bits |
| *Instruction Set* | 43 in basic instruction set; 17 optional instructions available with extended set (macroinstructions implemented by CPU-resident microprogram) |
| *Arithmetic* | Parallel, binary, fixed point, twos complement |

| | |
|---|---|
| *Memory* | 256 16-bit words of semiconductor read/write memory<br>Sockets for 512 16-bit words of semiconductor read-only memory<br>Capable of addressing 65,536 16-bit words |
| *Addressing* | Page size of 256 words. For direct and indirect modes:<br>● Absolute<br>● Relative to Program Counter<br>● Relative to Accumulator 2 (indexed)<br>● Relative to Accumulator 3 (indexed) |
| *Typical Instruction-Execution Speeds* | Register-to-register addition – 4.55 microseconds<br>Memory-to-register addition – 7.7 microseconds<br>Register input/output – 10.15 microseconds |
| *Input/Output and Control* | 16-bit data-input port<br>16-bit data-output bus<br>16-bit address bus<br>6 general-purpose flags<br>4 general-purpose jump-condition inputs<br>1 general interrupt input<br>1 control panel interrupt input |

## 1.3 POWER AND ENVIRONMENTAL REQUIREMENTS

The IMP-16C card contains both standard TTL integrated circuits and MOS/LSI components. These circuits require +5-volt and –12-volt 5% regulated dc supplies. In addition, the read/write memory on the card requires a –9-volt supply, which can be developed from the –12-volt source. Typical current requirements are given below for IMP-16C operation at maximum speed; however, when operating at lower speeds, the requirements on the –12-volt and –9-volt supplies are a few percent less than indicated below.

| | |
|---|---|
| *Voltage and Current* | +5 volts at 2.25 amperes<br>–12 volts at 0.5 amperes<br>–9 volts at 0.6 amperes |

When the IMP-16C on-card memory is disabled (see appendix C), the –9-volt power supply is not needed.

**NOTE**

Refer to 8.7 of this manual for operating procedures; these provide the prerequisite instructions for proper use of the IMP-16C.

| | |
|---|---|
| *Temperature* | Operating – 0 to 70° C<br>Storage – -20 to 100° C |
| *Humidity* | Maximum of 90% relative humidity without condensation |

# Chapter 2
# IMP-16C FUNCTIONAL DESCRIPTION

## 2.1 FUNCTIONAL UNITS AND DATA FLOW

A simplified block diagram showing the data flow among the major functional units of the IMP-16C is given in figure 2–1. The main components of the Central Processing Unit (CPU) and the Input/Output Section are shown. The Clock Generator and Timing Control, the Control Flags and Conditional Jump Multiplexer, and the Memory are not detailed at this time; they are discussed in detail in chapter 4.

### 2.1.1 INPUT DATA

Incoming data from peripheral units and data from memory are received by the Input Multiplexer and made available directly to the four Register and Arithmetic Logic Units (RALUs) and to the Data Buffer over a 16-bit data bus. A 4-bit bidirectional bus connects each RALU to the data bus.

### 2.1.2 DATA AND ADDRESS TRANSFERS

Both addresses and data are transferred from the Data Buffer over the 16-bit Buffered Data Out (BDO) bus. This bus is brought out to pins on the card-edge connector for transferring data to peripheral units and off-card memory. The on-card memory receives data directly from the BDO bus. Addresses are loaded into the Address Latches, from which they are brought out to another set of pins on the card-edge connector. Addresses are also routed to the on-card memory directly from the Address Latches.

### 2.1.3 CENTRAL PROCESSING UNIT (CPU) COMMUNICATIONS

Communications between the CROM(s) and the RALUs are effected over a 4-bit control bus. Because these units constitute the main part of the control and data processing capability of the IMP-16C, a comprehensive description of them follows.

Controlling the operations of the CPU is the Control Read-Only Memory (CROM). The control is effected by routines that constitute the microprogram stored in the Read-Only Memory of the CROM. The microprogram effects the implementation of macroinstructions that comprise the IMP-16C instruction set, with an expanded set available in a second CROM.

Because the actual data processing takes place in the four 4-bit Register and Arithmetic Logic Units (RALUs), they are described first (2.2) to establish a basis for their control by the Control Read-Only Memory (CROM) and also for the descriptions of the IMP-16C instruction set described in chapter 3.

## 2.2 REGISTER AND ARITHMETIC LOGIC UNITS (RALUs)

Four 4-bit RALUs constitute the Arithmetic Section shown in figure 2–2. This section includes the following major units:

- Input/Output Multiplexer
- Last-In/First-Out Stack (LIFOS)

- 16 Status Flags, storable and retrievable as a 16-bit register
  - Link Flag (L)
  - Carry Flag (CY)
  - Overflow Flag (OV)
  - 13 General-Purpose Flags (two of which are directly accessible to user at the edge connector)
- Program Counter (PC)
- Memory Data Register (MDR)
- Memory Address Register (MAR)
- 4 Accumulators (AC0, AC1, AC2, and AC3)
- Arithmetic and Logic Unit (ALU)
- Shifter
- Buses

The functions of these units are described in the following paragraphs.

### 2.2.1 INTERNAL BUSES

Three buses are internal to the RALU: A-bus, B-bus, and R-bus. These buses are described below.

A-Bus (Operand Bus). The contents of all RALU registers may be loaded onto the A-bus; data from the top of the Last-In/First-Out (LIFO) stack and from the RALU Status Flags (combined as a 16-bit word) may be loaded as operands. During such loading on the A-bus, the data may be complemented under control of the CROM. The contents of the A-bus may be gated through the ALU and Shifter to the R-bus or out of the RALU to the IMP-16C data bus through the Input/Output Multiplexer.

B-Bus (Operand Bus). The contents of all RALU registers may be loaded onto the B-bus. The contents of the B-bus may be loaded only into the Arithmetic and Logic Unit (ALU).

R-Bus (Result Bus). The R-bus serves to transfer the results of ALU operations to any RALU register, the LIFOS, and the RALU flags.

### 2.2.2 LAST-IN/FIRST-OUT STACK (LIFOS)

Each RALU has a stack that operates on a last-in/first-out basis. The stack is 16 words high and is accessible through the top location. The 16-bit-per-word stack of figure 2–2 is contained in the four 4-bit RALUs comprising the Arithmetic Section. A 16-bit data word is entered via the R-bus and retrieved via the A-bus. As a word is entered into the top location of the stack, the 16 bits in the top location are pushed down one level, and the entered bits occupy the top location. The contents of each lower level are replaced by the contents of the next higher location, and the contents of the bottom location are lost. The reverse process occurs when a word is retrieved from the stack; in this case, zeros are entered into the bottom location.

The stack is used primarily for saving of status during interrupts and for temporary storage of subroutine return addresses. It may also be used to temporarily store data using the appropriate instructions, described in chapter 3.

### 2.2.3 RALU FLAGS

There are 16 RALU status flags. These flags may be pushed onto the stack for temporary storage during interrupt processing. The flags may be transferred back into their respective flag flip-flops after completion of the interrupt service. Whenever the status flags are manipulated, the entire complement of flags is configured as a 16-bit register; the L (link), CY (carry), and OV (overflow) flags are the first, second, and third most significant bits, respectively, and the remaining general-purpose flags comprise the 13 less significant bits. The CY or the OV flags may be selected for output on the CYOV line from the RALUs, under control of the SEL control flag. For SEL control flag usage, see 4.3.

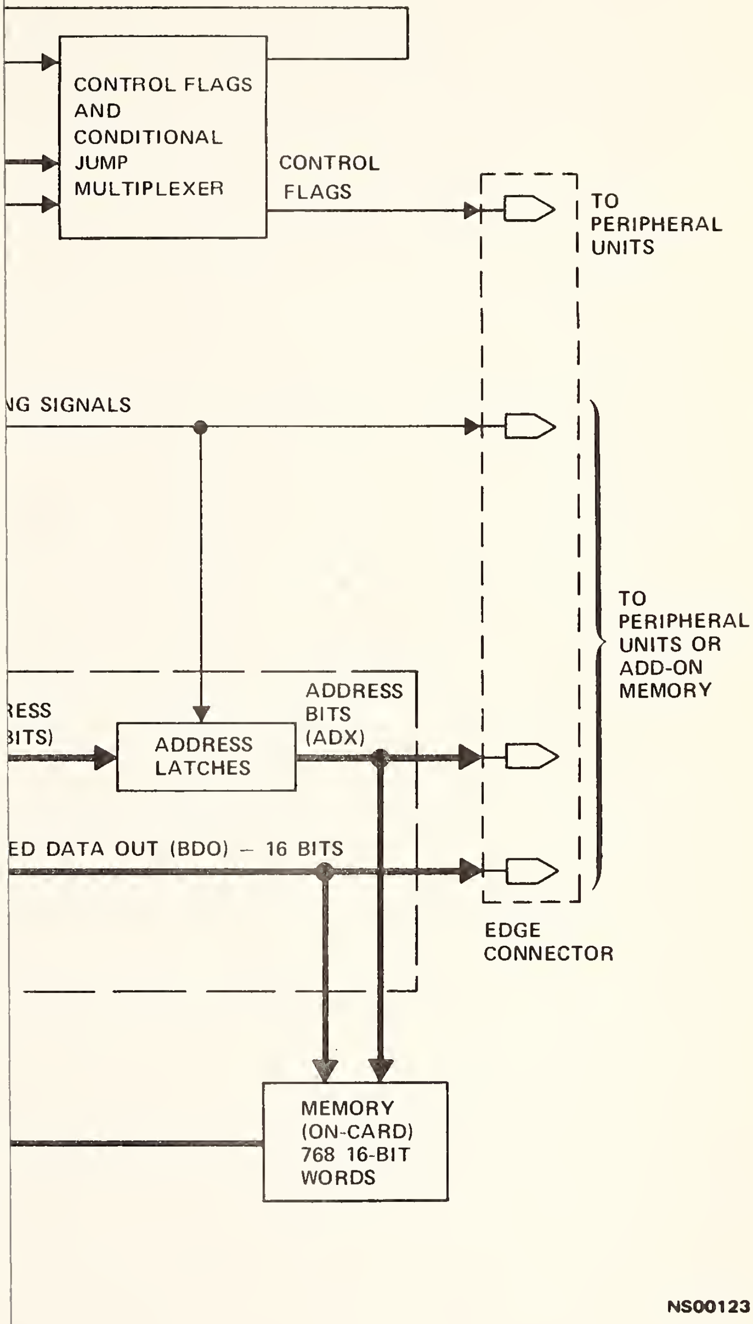
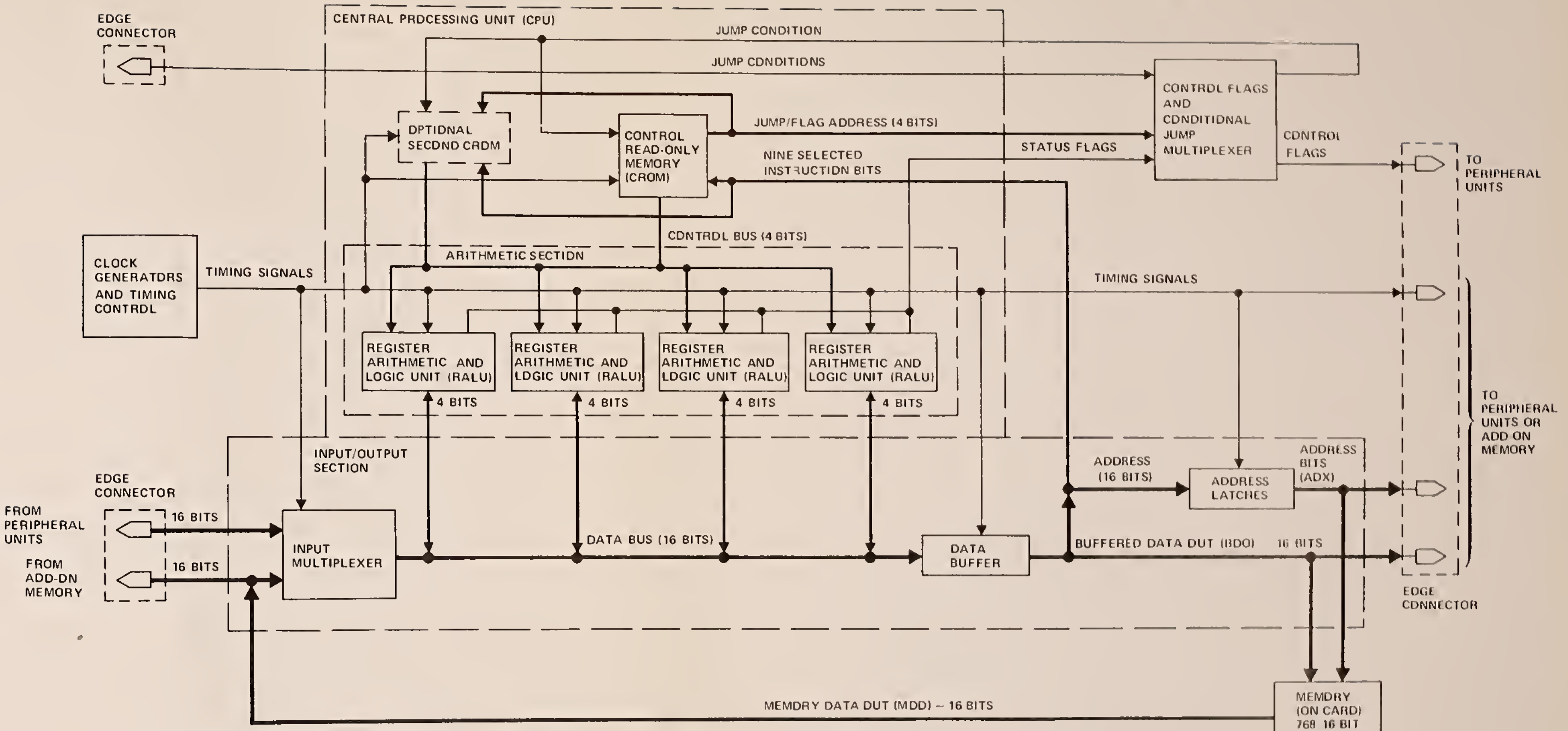

CONTROL FLAGS AND CONDITIONAL JUMP MULTIPLEXER
CONTROL FLAGS
TO PERIPHERAL UNITS
NG SIGNALS
TO PERIPHERAL UNITS OR ADD-ON MEMORY
RESS
BITS)
ADDRESS LATCHES
ADDRESS BITS (ADX)
ED DATA OUT (BDO) – 16 BITS
EDGE CONNECTOR
MEMORY (ON-CARD) 768 16-BIT WORDS
NS00123

*Figure 2–1. IMP-16C Simplified Block Diagram*


EDGE CONNECTOR
CENTRAL PROCESSING UNIT (CPU)
JUMP CONDITION
JUMP CONDITIONS
OPTIONAL SECOND CROM
CONTROL READ-ONLY MEMORY (CROM)
JUMP/FLAG ADDRESS (4 BITS)
NINE SELECTED INSTRUCTION BITS
STATUS FLAGS
CONTROL FLAGS AND CONDITIONAL JUMP MULTIPLEXER
CONTROL FLAGS
TO PERIPHERAL UNITS
CLOCK GENERATORS AND TIMING CONTROL
TIMING SIGNALS
CONTROL BUS (4 BITS)
ARITHMETIC SECTION
TIMING SIGNALS
REGISTER ARITHMETIC AND LOGIC UNIT (RALU)
REGISTER ARITHMETIC AND LOGIC UNIT (RALU)
REGISTER ARITHMETIC AND LOGIC UNIT (RALU)
REGISTER ARITHMETIC AND LOGIC UNIT (RALU)
4 BITS
4 BITS
4 BITS
4 BITS
TO PERIPHERAL UNITS OR ADD-ON MEMORY
INPUT/OUTPUT SECTION
EDGE CONNECTOR
FROM PERIPHERAL UNITS
16 BITS
FROM ADD-ON MEMORY
16 BITS
INPUT MULTIPLEXER
DATA BUS (16 BITS)
DATA BUFFER
ADDRESS (16 BITS)
ADDRESS LATCHES
ADDRESS BITS (ADX)
BUFFERED DATA OUT (BDO)
16 BITS
EDGE CONNECTOR
MEMORY DATA OUT (MDO) – 16 BITS
MEMORY (ON CARD) 768 16 BIT WORDS
N500123

NOTE:
THE SYMBOL ⊏> REPRESENTS A PIN (OR A GROUP OF PINS) ON THE IMP-16C CARD EDGE TERMINAL.

*Figure 2-1 IMP-16C Simplified Block Diagram*

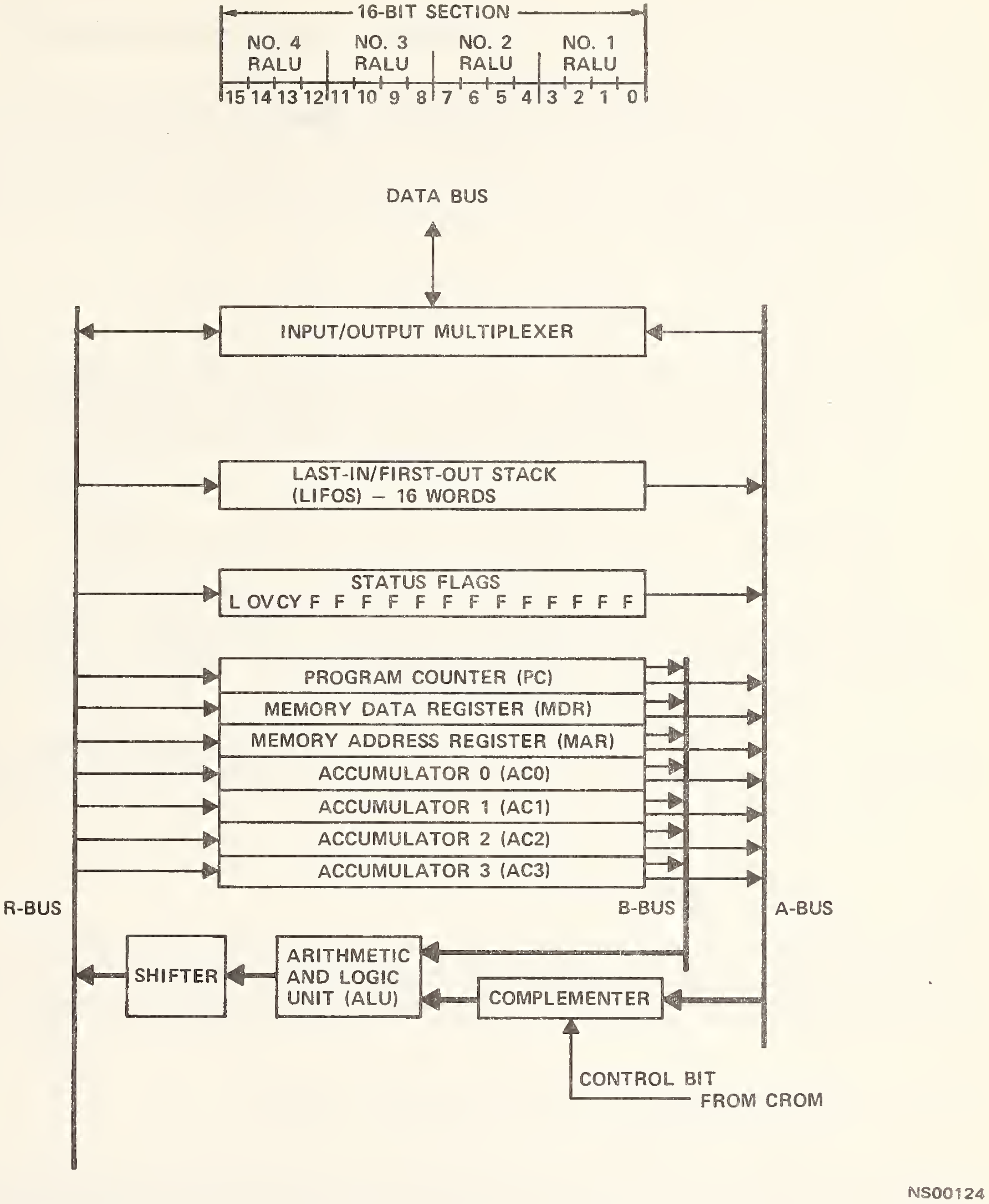


*Figure 2–2. IMP-16C 16-Bit Arithmetic Section*

### 2.2.4 PROGRAM COUNTER (PC)

The Program Counter (PC) holds the address of the next instruction to be executed. It is incremented by 1 immediately following the fetching of each instruction during execution of the current instruction. When there is a branch to another address in the main memory, the branch address is set into the Program Counter. A skip instruction merely increments the Program Counter by 1, thus causing the one instruction to be skipped.

### 2.2.5 MEMORY DATA REGISTER (MDR) AND MEMORY ADDRESS REGISTER (MAR)

The Memory Data Register (MDR) holds data transferred from the main memory to the processor, or vice versa. When fetching data, the effective address is placed in the Memory Address Register (MAR), and the fetch instruction causes the data word to be transferred from the designated main-memory location to the Memory Data Register. Conversely, when storing data in the main memory, the data word is placed in the Memory Data Register, the effective address is placed in the Memory Address Register, and the store instruction causes the data word to be transferred to the designated memory location.

### 2.2.6 ACCUMULATORS AC0, AC1, AC2, AND AC3

The accumulators hold operands for data manipulation during arithmetic and logical operations. Also, the result of an operation is usually stored temporarily in one of the four accumulators. Data words may be fetched from memory to the accumulator or stored from the accumulator into memory. The particular accumulator to take part in an operation is specified by the programmer in the appropriate instruction.

### 2.2.7 ARITHMETIC AND LOGIC UNIT (ALU), SHIFTER, AND COMPLEMENTER

The Arithmetic and Logic Unit (ALU) performs both arithmetic and logical operations: binary addition, AND, OR, and exclusive OR. Arithmetic and logical operations are effected by the microprogram that is part of the CROM. The IMP-16C performs arithmetic using the twos-complement technique. The contents of the A-bus may be selectively complemented under control of control bits from the CROM.

The output of the Arithmetic and Logic Unit is transferred to the R-bus through the Shifter and may then be stored in the stack or any of the RALU registers. The Shifter is capable of performing a single-position shift, either to the left or to the right, during each basic machine cycle.

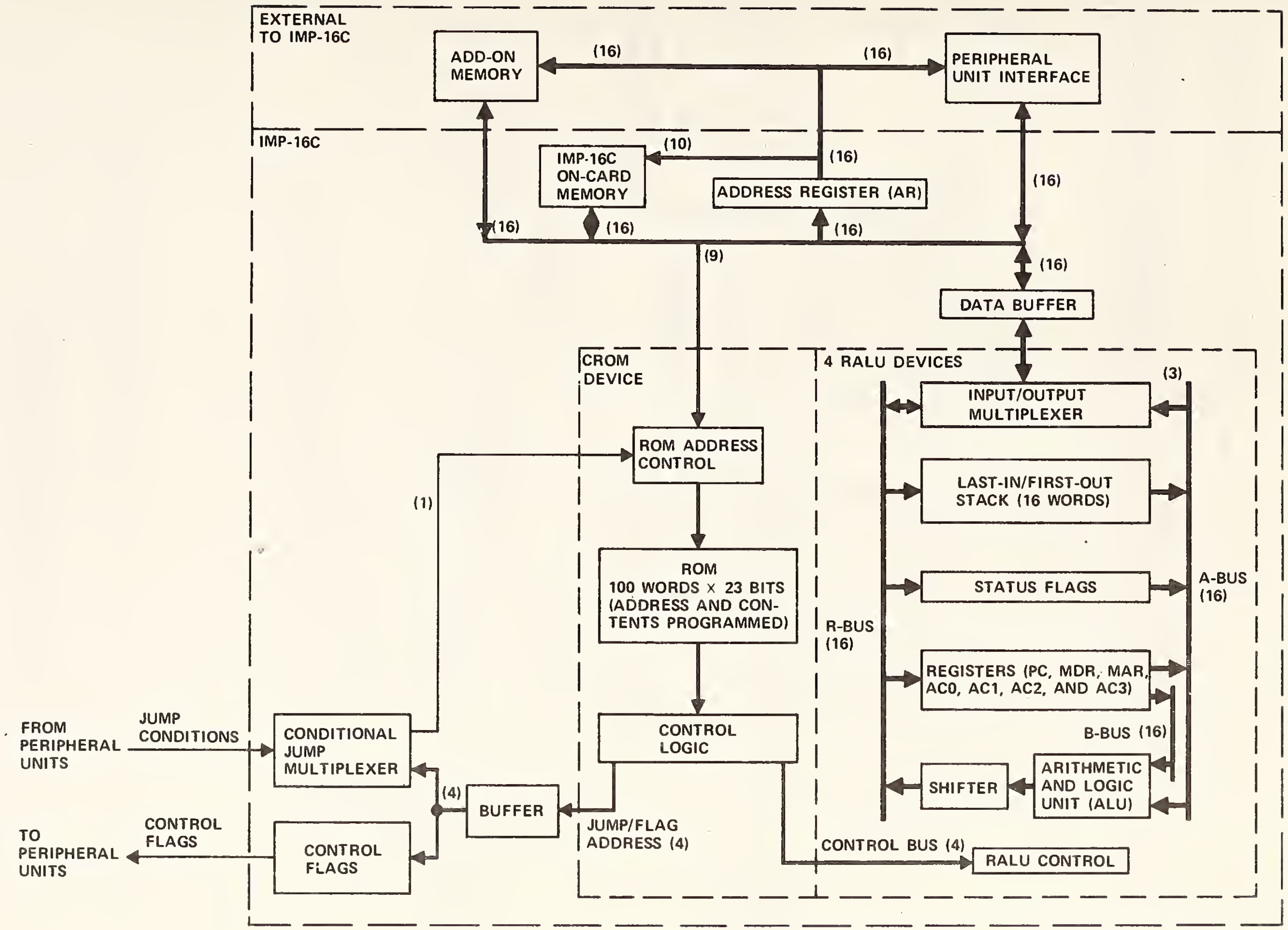
### 2.2.8 INPUT/OUTPUT MULTIPLEXER

The Input/Output Multiplexer handles data, address, and instruction transfers into and out of the RALU from or into the main memory and peripheral devices on a time-multiplexed basis. As shown in figure 2–2, output data (to data bus) must be received from the A-bus, and input data passes from the Input/Output Multiplexer to the R-bus.

## 2.3 CONTROL AND READ-ONLY MEMORY (CROM)

Figure 2–3 is a simplified block diagram emphasizing the role of the CROM and four RALUs as part of the IMP-16C. Data buffers are provided between the CPU MOS/LSI devices and other parts of the equipment that have been implemented with TTL logic. These buffers are TRI-STATE® logic elements that permit bus-connected inputs.

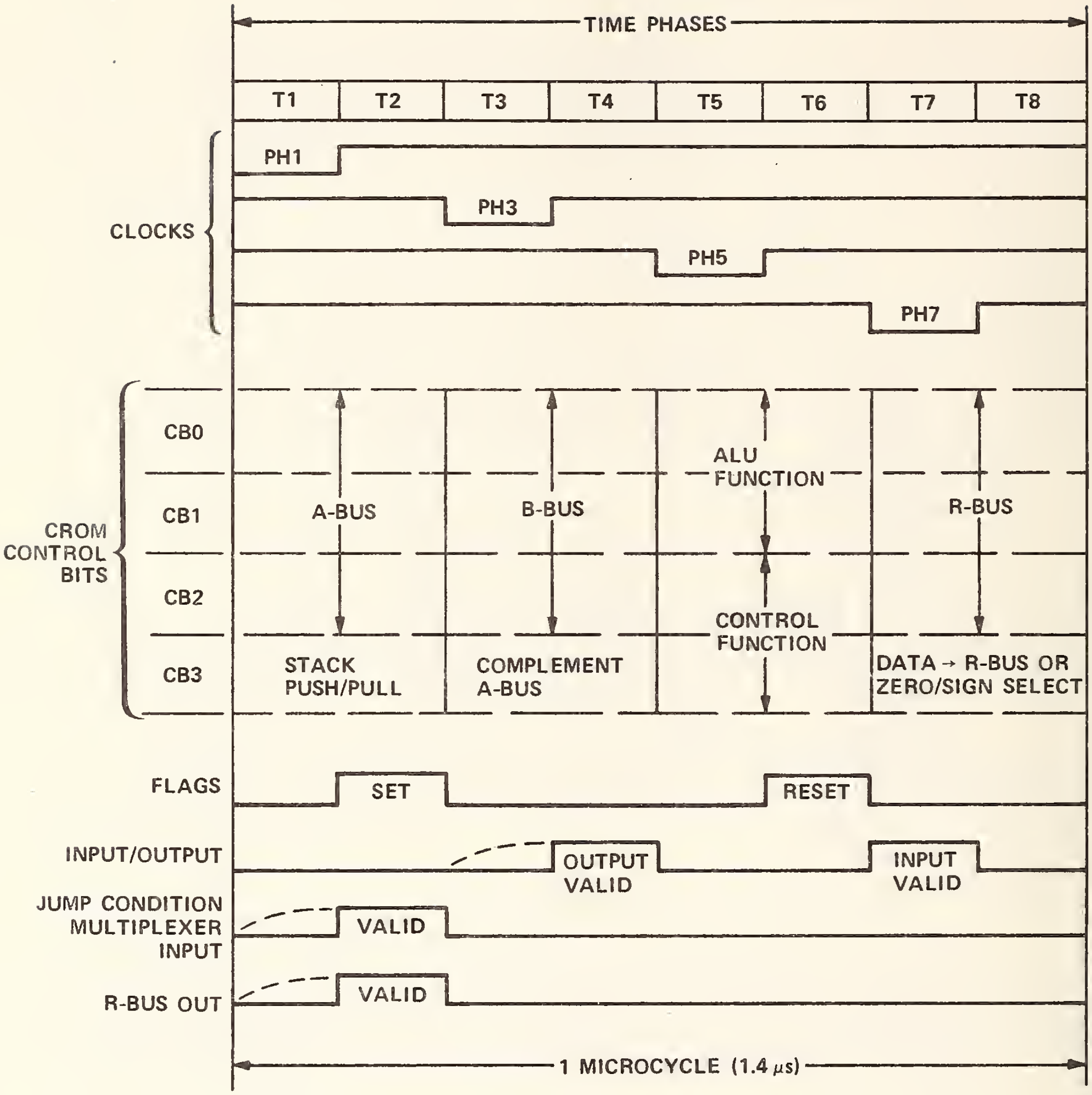
The primary control of the RALU devices is accomplished over the 4-bit control bus. The control bus is time-multiplexed to transfer four 4-bit words of control information per machine cycle. The functions effected by the control bits are indicated in figure 2–4.



NOTE: NUMBERS IN PARENTHESES DESIGNATE NUMBER OF LINES.

NS00125

*Figure 2–3. IMP-16C Control and Read-Only Memory (CROM) and Register and Arithmetic Logic Units (RALUs) Interrelations*



NS00126

*Figure 2–4. IMP-16C Timing Control*

The CROM contains a microprogram that implements the standard instruction set. This microprogram resides in a 100-by-23-bit ROM. During an instruction fetch, the 9 most significant bits of the instruction word are transferred to the CROM; these 9 bits comprise the opcode and other pertinent control fields of an instruction word. The instruction bits are decoded, and then the ROM Address Control in the CROM directs the control sequence to an entry point in the microprogram. The sequence continues until execution of the fetched instruction is completed. Then, the CROM goes through another fetch cycle to fetch the next instruction from memory. This process is continuously repeated.

## 2.4 IMP-16C TIMING

The basic machine cycle of the IMP-16C consists of the execution of a single microprogram step. This cyclic time period comprises eight time intervals: T1, T2,. . . ,T8. As indicated in the timing diagram of figure 2–4, clock pulses (phases 1, 3, 5, and 7) occur in the four clock lines at the respective odd-time periods T1, T3, T5, and T7.

### 2.4.1 EFFECT OF CONTROL BITS

**NOTE**

> The effects of the CROM control bits described in 2.4.1 are presented for completeness to show their relationship to the other signals in figure 2–4. Comprehension of this description is not required to understand the operation of the IMP-16C and, for this reason, may be deferred for later reading.

The encoding of the control bits shown in figure 2–4 for the 4-line time-multiplexed control bus (between the CROM and the RALU) is described below for each of the four time phases. It is during these time phases that control information is transferred from the CROM to the RALU. Control bus lines are designated CB0 through CB3.

**Phase 1 Control Bits**

- If the 3-bit A-bus address field is nonzero, the contents of the RALU register designated in the macroinstruction are gated onto the A-bus.
- If the A-bus address field is zero and the push-pull control bit is "1," the LIFO Stack is pulled, and the contents of the top of the stack are gated onto the A-bus.
- If the A-bus address field is zero and the push-pull control bit is "0," then a value of zero is gated onto the A-bus.

**NOTE**

> Pushing data onto the stack is also contingent on the push-pull control bit and is described under Phase 7 Control Bits, where the operation occurs.

**Phase 3 Control Bits**

- If the 3-bit B-bus address field is nonzero, then the contents of the register designated in the macroinstruction are gated onto the B-bus.
- If the B-bus address field is zero, a value of zero is gated onto the B-bus.
- The Complement A-bus bit causes the A input to the ALU to be complemented.

### Phase 5 Control Bits

- The ALU bits designate a function according to table 2–1.

*Table 2–1. ALU Function Bits*

| Code | Function |
|---|---|
| 00 | AND |
| 01 | XOR |
| 10 | OR |
| 11 | ADD |

- The Control Function bits designate a function according to table 2–2. The no-op function applies only to the Control Function field. The expression R ← 0/SIGN means that either zeros or the sign of the less significant byte of the word being transferred to the R-bus is propagated throughout the more significant byte. (If the fourth control bit CB3 is "1" during phase 7, the sign is propagated; otherwise, zero is propagated.)

*Table 2–2. Control Function Bits*

| Code | Function |
|---|---|
| 00 | No-Op (no operation for control bits only) |
| 01 | R ← 0/SIGN (zero or sign propagation) |
| 10 | LSH (Left Shift) |
| 11 | RSH (Right Shift) |

### Phase 7 Control Bits

- If the 3-bit R-bus field is nonzero, the contents of the R-bus are gated into the register addressed by this field.

- If the Control Function bits transferred during phase 5 do not specify R←0/SIGN (see table 2–2), then CB3 specifies the source of the data gated onto the R-bus:
  - If CB3 is "0," the source is the output of the ALU.
  - If CB3 is "1," the data comes from an external source via the Data Multiplexer.

- If R ← 0/SIGN is specified, then CB3 specifies whether zero or the sign is propagated throughout the more significant byte.

- If the R-bus address is zero and the push-pull control bit was active (during phase 1), a data word is pushed onto the LIFO Stack from the R-bus.

### 2.4.2 MISCELLANEOUS TIMING SIGNALS

The control of the Flag Flip-Flops is indicated on the timing diagram (figure 2–4). A unique flag address is established during each machine cycle; at T2 the flag may be set, and/or at T6 the flag may be reset. It is thus possible to set, reset, or pulse a flag during a single machine cycle.

The Conditional Jump Multiplexer shares the same address lines as the Flag Flip-Flops. If a conditional jump is being performed, the condition is tested during T2.

### 2.4.3 DATA TRANSFER TIMING

Data transfers between the RALU and the Data Bus may occur at three times during each microcycle:

- During T4, the data word presently on the A-bus is gated onto the Data Bus. The information that appears on the Data Bus at this time either is output data destined for memory or an external device, or is an address value used for loading the Address Register latch (shared by memory and external devices).
- At T2, the contents of the R-bus (result of preceding microcycle) are gated onto the Data Bus. Primarily, the data provide inputs to the Conditional Jump Multiplexer to permit testing of the result of the operation performed on the previous microcycle.
- During T7, data to be transferred to the RALU appear on the Data Bus. The data may then be gated onto the R-bus by the RALU's Input/Output Multiplexer and, subsequently, may be stored in one of the working registers (AC0 through AC3).

## 2.5 IMP-16C OPERATION

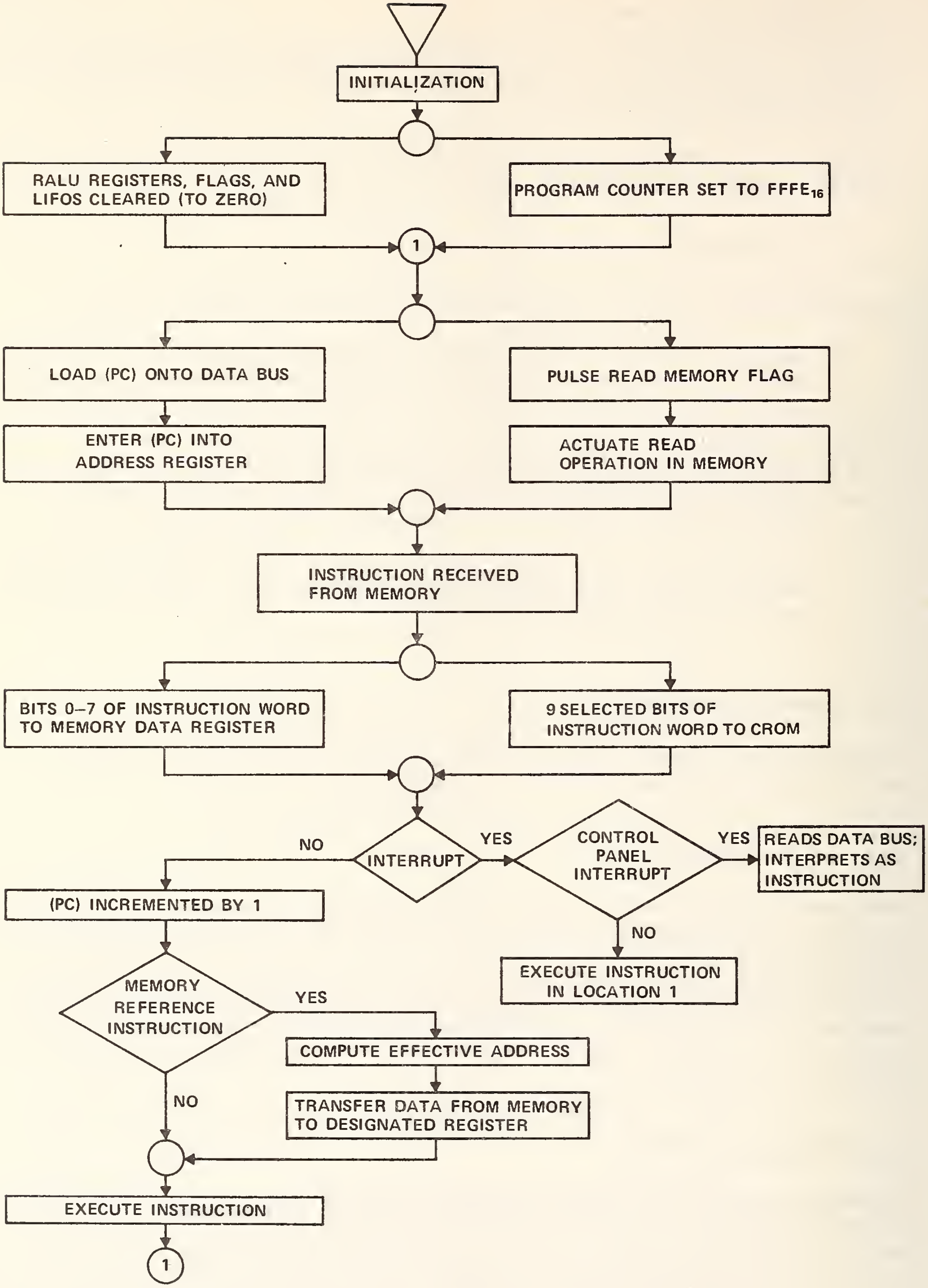
A flowchart of the events that occur during a typical operation of the IMP-16C is given in figure 2–5. The various events are further elaborated upon in the following paragraphs.

### 2.5.1 INITIALIZATION

When power is first applied to the processor, all RALU registers, flags, and the stack are cleared to zero. The microprogram then enters an initialization sequence, in which the Program Counter (PC) is set to a starting value of $FFFE_{16}$ (the next-to-last location in the top page of memory, assuming a maximum of $2^{16}$ memory locations). The reasons for choosing this location over location $0000_{16}$ or any other location can be explained as follows. In most applications, the first few executable statements in a macroprogram are usually initialize routines and other program segments of a supervisory nature. By keeping these in the top portion of memory, the supervisor programs can be stored in ROM for nonvolatility. Since the base page is directly accessible from anywhere in memory, it is desirable for it to be implemented with read/write memory.

### 2.5.2 INSTRUCTION FETCH

Following the initialize sequence, control is transferred to the first step in the fetch microroutine. During this first microcycle of the fetch routine, the contents of the Program Counter (PC) are sent out on the data bus at T4; at the same time, the Read Memory Flag is set high. This causes the external Address Register to be loaded with the contents of the PC. The Read Memory Flag actuates a read operation in the system memory. During T7 of the same microcycle, the 16-bit instruction comes back from memory ready to be decoded. At that time, bits 0 through 7 are placed in the RALU Memory Data Register, and bits 7 through 15 are loaded into the CROM. Bits 8 through 15 of the Memory Data Register are set equal to the value of bit 7.


INITIALIZATION
RALU REGISTERS, FLAGS, AND LIFOS CLEARED (TO ZERO)
PROGRAM COUNTER SET TO $FFFE_{16}$
1
LOAD (PC) ONTO DATA BUS
PULSE READ MEMORY FLAG
ENTER (PC) INTO ADDRESS REGISTER
ACTUATE READ OPERATION IN MEMORY
INSTRUCTION RECEIVED FROM MEMORY
BITS 0–7 OF INSTRUCTION WORD TO MEMORY DATA REGISTER
9 SELECTED BITS OF INSTRUCTION WORD TO CROM
NO
INTERRUPT
YES
CONTROL PANEL INTERRUPT
YES
READS DATA BUS; INTERPRETS AS INSTRUCTION
NO
EXECUTE INSTRUCTION IN LOCATION 1
(PC) INCREMENTED BY 1
MEMORY REFERENCE INSTRUCTION
YES
COMPUTE EFFECTIVE ADDRESS
TRANSFER DATA FROM MEMORY TO DESIGNATED REGISTER
NO
EXECUTE INSTRUCTION
1

NS00127

*Figure 2–5. IMP-16C Instruction Execution Flowchart*

In the next microcycle, the Program Counter is incremented to point to the next consecutive memory location.

A decision is made as to whether the instruction just read in needs to make reference to memory. If it does not (for register-register operations and other nonmemory reference instructions), then the instruction decode circuit steers control to a microprogram entry point that corresponds to the particular instruction. If the instruction requires a memory reference, then two additional microprogram steps are required to compute an effective address and to bring in the new word: first, the memory address is computed, and, second, the data word is transferred from memory to the register designated by the preceding instruction.

### 2.5.3 COMMUNICATIONS WITH MEMORY

During a memory-read operation, the microprogram sends an address on the data bus at T4; this address is loaded into the Address Register under control of the Read Memory Flag. Similarly, during a write-memory operation, the address is loaded under control of the Load Address Flag. These flags are further discussed in chapter 3.

When executing a memory-read operation, the processor sends out an address on the bus at T4 and expects data back at T7 of the same microcycle. A circuit, described in chapter 4, is included on the IMP-16C card to extend the interval between T4 and T7 to accommodate memories whose access times are longer than the normal T4-to-T7 interval.

A memory-write operation takes 2 microcycles, because both address and data have to be sent out. The address is sent out during T4 of the first microcycle and is latched in the address register. The output data destined for memory arrives during T4 of the next microcycle and can be used directly to write memory. In the IMP-16C, T4 is stretched for two additional periods to accommodate both the read and write delays necessary to communicate with the system memory.

### 2.5.4 EXECUTION OF INSTRUCTIONS

The CROM in the IMP-16C contains a microprogram that implements the standard instruction set. Each macroinstruction in a user's program is brought into the processor under control of the CROM's microprogram instruction fetch routine. This instruction is then decoded, and the ROM Address Control in the CROM directs the control sequence to an entry point in the microprogram. The sequence continues until execution is completed. Then, the CROM goes through another instruction-fetch cycle to bring in the next macroinstruction. The process is repeated continuously until directed otherwise by a macroinstruction such as Halt or an interrupt condition.

A test for an interrupt condition is made at the beginning of each instruction fetch. If the interrupt line is high, the CROM transfers control to a microprogram sequence that determines the type of interrupt that occurred. If a control panel interrupt is not active, the CROM assumes that a general interrupt is in effect. For a general interrupt, the contents of the PC are saved on the stack, and then the Interrupt Routine in memory location 1 is executed; otherwise, the Control Panel Subroutine is executed.

# Chapter 3
# IMP-16C INSTRUCTION SET

## 3.1 INTRODUCTION

Eight functional types of instructions comprise the basic IMP-16C instruction set:

- Load and Store
- Arithmetic
- Logical
- Skip
- Shifts
- Transfer of Control
- Register
- Input/Output and Miscellaneous

The instructions for each functional type are described as a group. For each instruction, the name of the instruction, its mnemonic, its word format, its operation in the form of an equation, and a succinct explanation of its operation are given. A tabulated summary of each type of instruction precedes the detailed descriptions. An extended instruction set (available as an option) is described in 3.7.

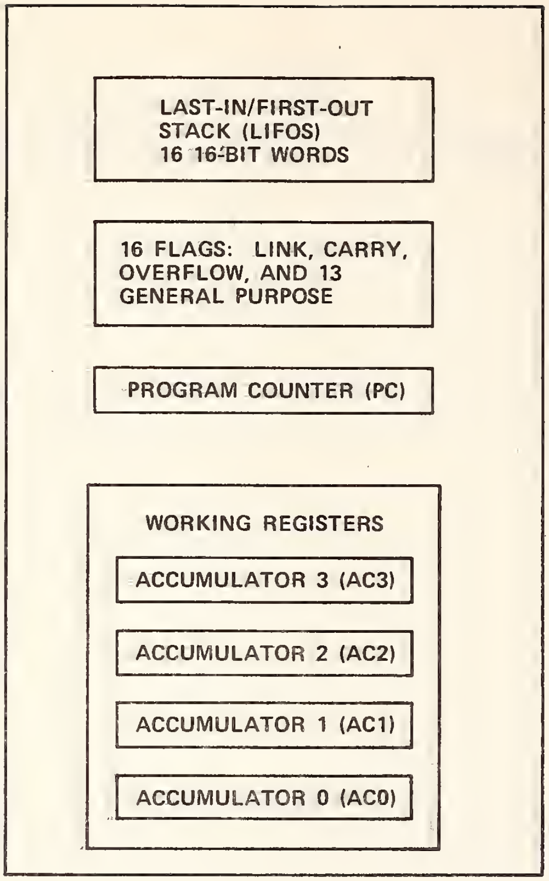
Before describing the instructions, brief descriptions of the registers referred to in the instruction descriptions are given.

The IMP-16C instructions and their assembler language opcode mnemonics are summarized in appendix A.

## 3.2 ARITHMETIC AND LOGIC UNITS REFERENCED IN IMP-16C INSTRUCTIONS

The units referenced in the ensuing description of the IMP-16C instructions are listed below and are shown in block diagram form in figure 3–1.

- Last-In/First-Out Stack (LIFOS)
- Register and Arithmetic Logic Unit (RALU) Flags
- Program Counter (PC)
- Accumulator 0 (AC0)
- Accumulator 1 (AC1)
- Accumulator 2 (AC2)
- Accumulator 3 (AC3)



NS00128

*Figure 3–1. Arithmetic and Logic Units Referenced in IMP-16C Instructions*

### 3.2.1 LAST-IN/FIRST-OUT STACK (LIFOS)

The IMP-16C has a hardware stack that data may be stored in or retrieved from on a last-in/first-out basis. The stack is 16 words deep and is accessible through the top location. As a data word is entered into the stack, the contents of the top location and each other location are pushed downward to the next lower level; if the stack is full, the word in the bottom location is lost. Conversely, the contents of the top location are pulled from the stack during retrieval of a data word; the top location and each lower location are replaced by the contents of the next lower location, and zeros are entered into the bottom location.

The stack is used primarily for saving status during interrupts and for saving subroutine return addresses. It may be used also for temporary storage of data using the PUSH, PULL, XCHRS, PUSHF, and PULLF instructions (described later in this chapter).

### 3.2.2 REGISTER AND ARITHMETIC LOGIC UNIT (RALU) FLAGS

There are 16 RALU status flags. These flags may be pushed onto the stack (saved) or may be loaded from the stack (restored). During such operations, the flags are configured as a 16-bit word: the L (Link), CY (Carry), and OV (Overflow) flags are the first, second, and third most significant bits, respectively, and the remaining 13 general-purpose flags comprise the remaining 13 less significant bits (figure 3–11).

Flags 0 and 12 are externally available. The L flag is primarily used in some shifting operations, and the CY and OV flags are adjuncts for arithmetic operations. The specific uses of the flags are elaborated upon in the appropriate instruction descriptions.

### 3.2.3 PROGRAM COUNTER (PC)

The Program Counter (PC) holds the address of the next instruction to be executed. When there is a branch to another address in the main memory, the branch address is set into the Program Counter. A skip instruction merely increments the Program Counter by 1, thus causing the one instruction to be skipped.

### 3.2.4 ACCUMULATORS 0, 1, 2, AND 3 (AC0, AC1, AC2, AND AC3)

The accumulators are used as working registers for data manipulation. Data may be fetched from a location in memory to an accumulator, and may be stored from an accumulator to a location in memory. The particular accumulator to take part in an operation is specified by the programmer in the appropriate instruction.

## 3.3 DATA AND INSTRUCTIONS

### 3.3.1 DATA REPRESENTATION

Data are represented in the IMP-16C in twos-complement integer notation. In this system, the negative of a number is formed by complementing each bit in the data word and adding 1 to the complemented number. The sign is indicated by the most significant bit. In the 16-bit word of the IMP-16C, when bit 15 is a "0," it denotes a positive number; when it is a "1," it denotes a negative number. Maximum number ranges for this system are $7FFF_{16}$ ($+32767_{10}$) and $8000_{16}$ ($-32768_{10}$). The carry flag is set to the value of the most significant bit position (bit 15) resulting from an add operation. The overflow flag is set to "1" if two numbers having like signs are added and the sign of the resulting sum is different from that of the operands.

### 3.3.2 INSTRUCTIONS

There are eight classes of IMP-16C instructions. Each class of instruction and the associated instructions are summarized in a table preceding the descriptions of the instructions. Also, the applicable instruction word formats are defined.

## 3.4 MEMORY ADDRESSING

The IMP-16C instruction set provides for direct and indirect memory addressing. For direct addressing, three distinct modes are available: base page (or absolute), program-counter relative, and indexed. The mode of addressing is specified by the XR field of the simplified instruction word format shown in figure 3–2.

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OPERATION CODE | | | | | | XR FIELD | | DISPLACEMENT FIELD | | | | | | | |

*Figure 3–2. Instruction Word for Addressing Memory*

### 3.4.1 BASE PAGE ADDRESSING

When the XR field is 00, it specifies base page addressing. Base page is directly accessible from any location in the address space of the memory. In this mode, the effective address is formed by setting bits 8 through 15 to zero and using the value of the 8-bit displacement field as an absolute address. Up to 256 words (locations 0 through 255) may be addressed in this way.

### 3.4.2 PROGRAM-COUNTER RELATIVE ADDRESSING

Program-counter relative addressing is specified when the XR field is 01. The displacement is treated as a signed number such that its sign bit (bit 7) is propagated to bits 8 through 15, and the effective address is formed by adding the contents of the PC to the resulting number. This permits PC-relative addressing −128 and +127 locations from the PC value; however, at the time of formation of the address, the PC has already been incremented in the microprogram and is pointing to the next macroinstruction. Because of this, the actual addressing range is from −127 to +128 from the current instruction.

### 3.4.3 INDEXED ADDRESSING

Indexed addressing is done with reference to Accumulator 2 or 3 (AC2 or AC3). In this mode, the displacement field is again interpreted as a signed 8-bit number from −128 to +127 with the sign (bit 7) extended through bits 8 through 15. The contents of the chosen index register (AC2 when $xr = 10_2$ and AC3 when $xr = 11_2$) are added to the number formed from the displacement value to yield an effective address that can reach any location in 65,536 words of memory.

*Table 3–1. Summary of Addressing Modes*

| XR Field | Addressing Mode | Effective Address | Range |
|---|---|---|---|
| 00 | Base | EA = disp | $0 \leqslant disp \leqslant 255$ |
| 01 | Relative to Program Counter | EA = disp + (PC) | $-128 \leqslant disp \leqslant 127$ |
| 10 | Relative to AC2 | EA = disp + (AC2) | $-128 \leqslant disp \leqslant 127$ |
| 11 | Relative to AC3 | EA = disp + (AC3) | $-128 \leqslant disp \leqslant 127$ |

### 3.4.4 INDIRECT ADDRESSING

Indirect addressing is accomplished by first calculating the effective address (EA) using the same method used for direct addressing; the memory location at this address contains a number that is then used as the address of the operand. The following instructions use indirect addressing:

- Load Indirect (see table 3–3)
- Store Indirect (see table 3–3)
- Jump Indirect (see table 3–7)
- Jump to Subroutine Indirect (see table 3–7)

## 3.5 NOTATION AND SYMBOLS USED IN IMP-16C INSTRUCTION DESCRIPTIONS

Refer to table 3–2 for definitions of the notation and symbols used in the IMP-16C instruction descriptions. The notations are given first in alphabetical order followed by the symbols. Upper-case mnemonics refer to fields in the instruction word; lower-case mnemonics refer to the numerical value of the corresponding fields. In cases where both lower- and upper-case mnemonics are composed of the same letters, only the lower-case mnemonic is given in table 3–2. The use of lower-case notation designates variables.

*Table 3–2. Notation Used in Instruction Descriptions*

| Notation | Meaning |
|---|---|
| ACr | Denotes a specific working register (AC0, AC1, AC2, or AC3), where *r* is the number of the accumulator referenced in the instruction. |
| AR | Denotes the address register used for addressing memory or peripheral devices. |
| cc | Denotes the 4-bit condition code value for conditional branch instructions. |
| ctl | Denotes the 7-bit control-field value for flag, input/output, and miscellaneous instructions. |
| CY | Indicates that the Carry flag is set if there is a carry due to the instruction (either an addition or a subtraction). |
| disp | Stands for displacement value and it represents an operand in a nonmemory reference instruction or an address field in a memory reference instruction. It is an 8-bit, signed twos-complement number except when base page is referenced; in the latter case, it is unsigned. |
| dr | Denotes the number of a destination working register that is specified in the instruction-word field. The working register is limited to one of four: AC0, AC1, AC2, or AC3. |
| EA | Denotes the effective address specified by the instruction directly, indirectly, or by indexing. The contents of the effective address are used during execution of an instruction. See table 3–1. |
| fc | Denotes the number of the referenced flag (see table 3–13 under 3.6.8, Input/Output, Halt, and Flag Instructions). |
| INTEN | Denotes the Interrupt Enable control flag. |
| IOREG | Denotes an input/output register in a peripheral device. |
| L | Denotes 1-bit link (L) flag. |
| OV | Indicates that the overflow flag is set if there is an overflow due to the instruction (either an addition or a subtraction). |
| PC | Denotes the program counter. During address formation, it is incremented by 1 to contain an address 1 greater than that of the instruction being executed. |
| PC | Denotes the number of a working register that is specified in the instruction-word field. The working register is limited to one of four: AC0, AC1, AC2, or AC3. |
| SEL | Denotes the Select control flag. It is used to select the carry or overflow for output on the carry and overflow (CYOV) line of the CPU, and to include the link bit (L) in shift operations. |
| sr | Denotes the number of a source working register that is specified in the instruction-word field. The working register is limited to one of four: AC0, AC1, AC2, or AC3. |
| xr | When not zero, this value designates the number of the register to be used in the indexed and relative memory-addressing modes. See table 3–1. |

*Table 3–2. Notation Used in Instruction Descriptions (Continued)*

| Notation | Meaning |
|---|---|
| ( ) | Denotes the contents of the item within the parentheses. (ACr) is read as "the contents of ACr." (EA) is read as "the contents of EA." |
| [ ] | Denotes "the result of." |
| ~ | Indicates the logical complement (ones complement) of the value on the right-hand side of ~. |
| → | Means "replaces." |
| ← | Means "is replaced by." |
| @ | Appearing in the operand field of an instruction, denotes indirect addressing. |
| ∧ | Denotes an AND operation. |
| ∨ | Denotes an OR operation. |
| ⊽ | Denotes an exclusive OR operation. |

## 3.6 INSTRUCTION DESCRIPTIONS

Each class and subclass of instruction is introduced by a table that lists and summarizes the instructions. The word format then is illustrated. Detailed descriptions are given, providing the following information:

- Name of instruction followed by operation code mnemonic in parentheses
- Operation Code in word format diagram
- Operation in equation notation
- Description of operation in detail

### 3.6.1 LOAD AND STORE INSTRUCTIONS

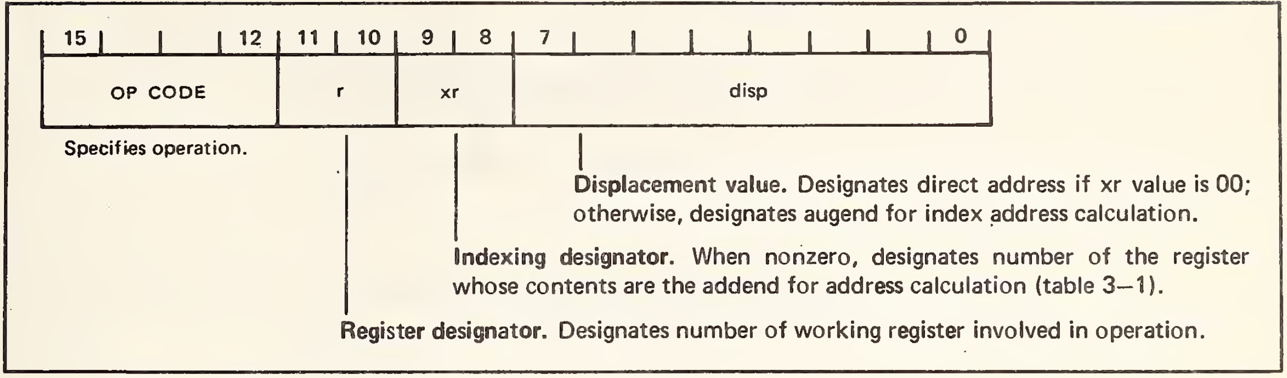
There are four instructions in this group. These are summarized in table 3–3 and then individually described. The word format is shown in figure 3–3.

*Table 3–3. Load and Store Instructions*

| Instruction | OpCode | Operation | Assembler Format |
|---|---|---|---|
| LOAD | 1000 | (ACr) ← (EA) | LD r, disp(xr) |
| LOAD INDIRECT | 1001 | (ACr) ← ((EA)) | LD r, @disp(xr) |
| STORE | 1010 | (EA) ← (ACr) | ST r, disp(xr) |
| STORE INDIRECT | 1011 | ((EA)) ← (ACr) | ST r, @disp(xr) |

NOTE

For indirect operations, the symbol @ must precede the memory location designated in the operand field of the assembler instruction.



Specifies operation.

Displacement value. Designates direct address if xr value is 00; otherwise, designates augend for index address calculation.

Indexing designator. When nonzero, designates number of the register whose contents are the addend for address calculation (table 3–1).

Register designator. Designates number of working register involved in operation.

*Figure 3–3. Load and Store Instruction Format*

**Load (LD)**

| 15 | | | 12 | 11 | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | r | | xr | | disp | | | | | | | |

*Operation:* (ACr) ← (EA)

*Description:* The contents of ACr are replaced by the contents of EA. The initial contents of ACr are lost; the contents of EA are unaltered.

**Load (LD) Indirect**

| 15 | | | 12 | 11 | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | r | | xr | | disp | | | | | | | |

*Operation:* (ACr) ← ((EA))

*Description:* The contents of ACr are replaced indirectly by the contents of EA. The initial contents of ACr are lost; the contents of EA and the location that designates EA are unaltered.

Store (ST)

| 15 | | | 12 | 11 | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | r | | xr, | | disp | | | | | | | |

*Operation:* (EA) ← (ACr)

*Description:* The contents of EA are replaced by the contents of ACr. The initial contents of EA are lost; the contents of ACr are unaltered.

Store (ST) Indirect

| 15 | | | 12 | 11 | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | r | | xr | | disp | | | | | | | |

*Operation:* ((EA)) ← (ACr)

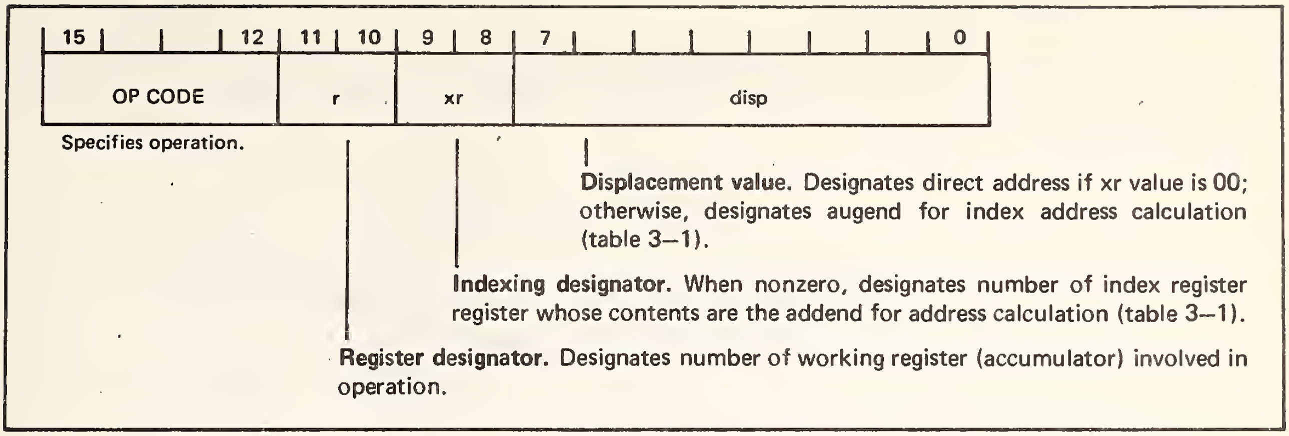
*Description:* The contents of EA are replaced indirectly by ACr. The initial contents of EA are lost; the contents of ACr and the location that designates EA are unaltered.

### 3.6.2 ARITHMETIC INSTRUCTIONS

There are two instructions in this group summarized in table 3–4 and then described individually. Either of these instructions may be carried out with any of the four general-purpose accumulators (AC0, 1, 2, or 3). The word format is shown in figure 3–4.

*Table 3–4. Arithmetic Instructions*

| Instruction | OpCode | Operation | Assembler Format |
|---|---|---|---|
| ADD (ADD) | 1100 | (ACr) ← (ACr) + (EA),OV,CY | ADD r, disp(xr) |
| SUBTRACT (SUB) | 1101 | (ACr) ← (ACr) + ~ (EA) + 1,OV,CY | SUB r, disp(xr) |



Specifies operation.

**Displacement value.** Designates direct address if xr value is 00; otherwise, designates augend for index address calculation (table 3–1).

**Indexing designator.** When nonzero, designates number of index register register whose contents are the addend for address calculation (table 3–1).

**Register designator.** Designates number of working register (accumulator) involved in operation.

*Figure 3–4. Arithmetic Instruction Format*

### Add (ADD)

| 15 — 12 | 11 10 | 9 8 | 7 — 0 |
|---|---|---|---|
| 1 1 0 0 | r | xr | disp |

*Operation:* (ACr) ← (ACr) + (EA),OV,CY

*Description:* The contents of ACr are added algebraically to the contents of the effective memory location EA. The sum is stored in ACr, and the contents of EA are unaltered. The preceding contents of ACr are lost. The carry and overflow flags are set according to the result of the operation.

### Subtract (SUB)

| 15 — 12 | 11 10 | 9 8 | 7 — 0 |
|---|---|---|---|
| 1 1 0 1 | r | xr | disp |

*Operation:* (ACr) ← (ACr) + ~ (EA) + 1,OV,CY

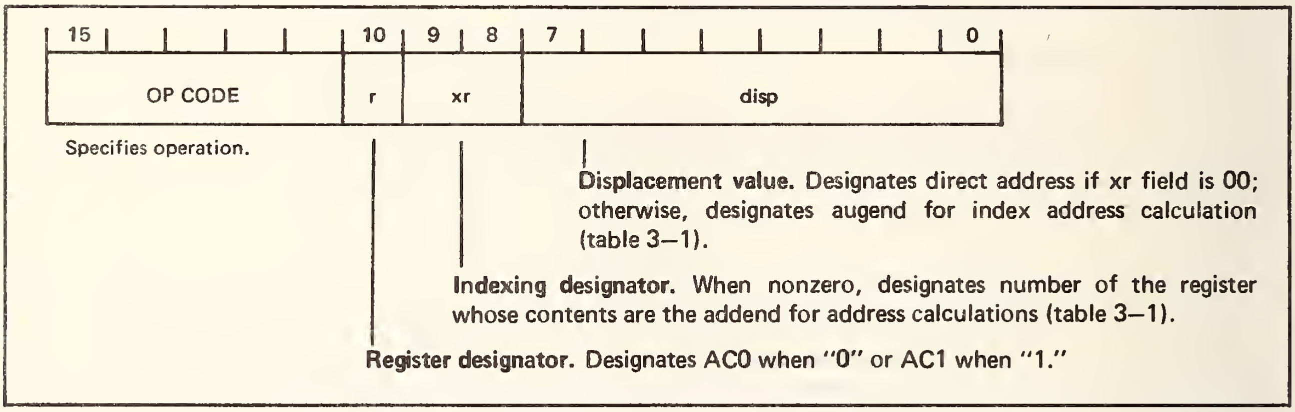
*Description:* The contents of ACr are added to the twos complement of the effective memory location EA. The result is stored in ACr, and the effective memory location is unaltered. The carry and overflow flags are set according to the result of the operation.

### 3.6.3 LOGICAL INSTRUCTIONS

There are two instructions in this group, summarized in table 3–5 and then described individually. Either of these instructions may be carried out with only two of the general-purpose accumulators, AC0 or AC1. The word format is shown in figure 3–5.

*Table 3–5. Logical Instructions*

| Instruction | OpCode | Operation | Assembler Format |
|---|---|---|---|
| AND | 01100 | (ACr) ← (ACr) ∧ (EA) | AND r, disp(xr) |
| OR | 01101 | (ACr) ← (ACr) ∨ (EA) | OR r, disp(xr) |



Specifies operation.

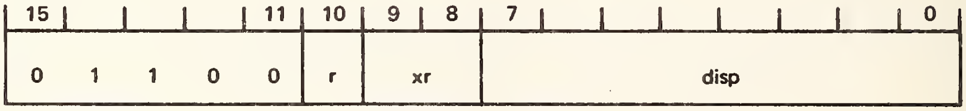
Displacement value. Designates direct address if xr field is 00; otherwise, designates augend for index address calculation (table 3–1).

Indexing designator. When nonzero, designates number of the register whose contents are the addend for address calculations (table 3–1).

Register designator. Designates AC0 when "0" or AC1 when "1."

*Figure 3–5. Logical Instruction Format*

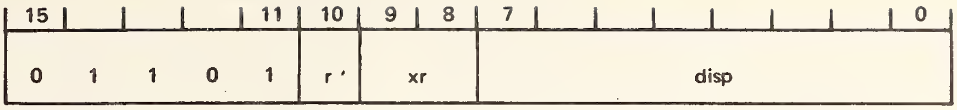
And (AND)



*Operation:* (ACr) ← (ACr) ∧ (EA)

*Description:* The contents of ACr (where r is either 0 or 1) and the contents of the effective memory location EA are ANDed, and the result is stored in ACr. The initial contents of ACr are lost, and the contents of EA are unaltered.

**Or(OR)**



*Operation:* (ACr) ← (ACr) ∨ (EA)

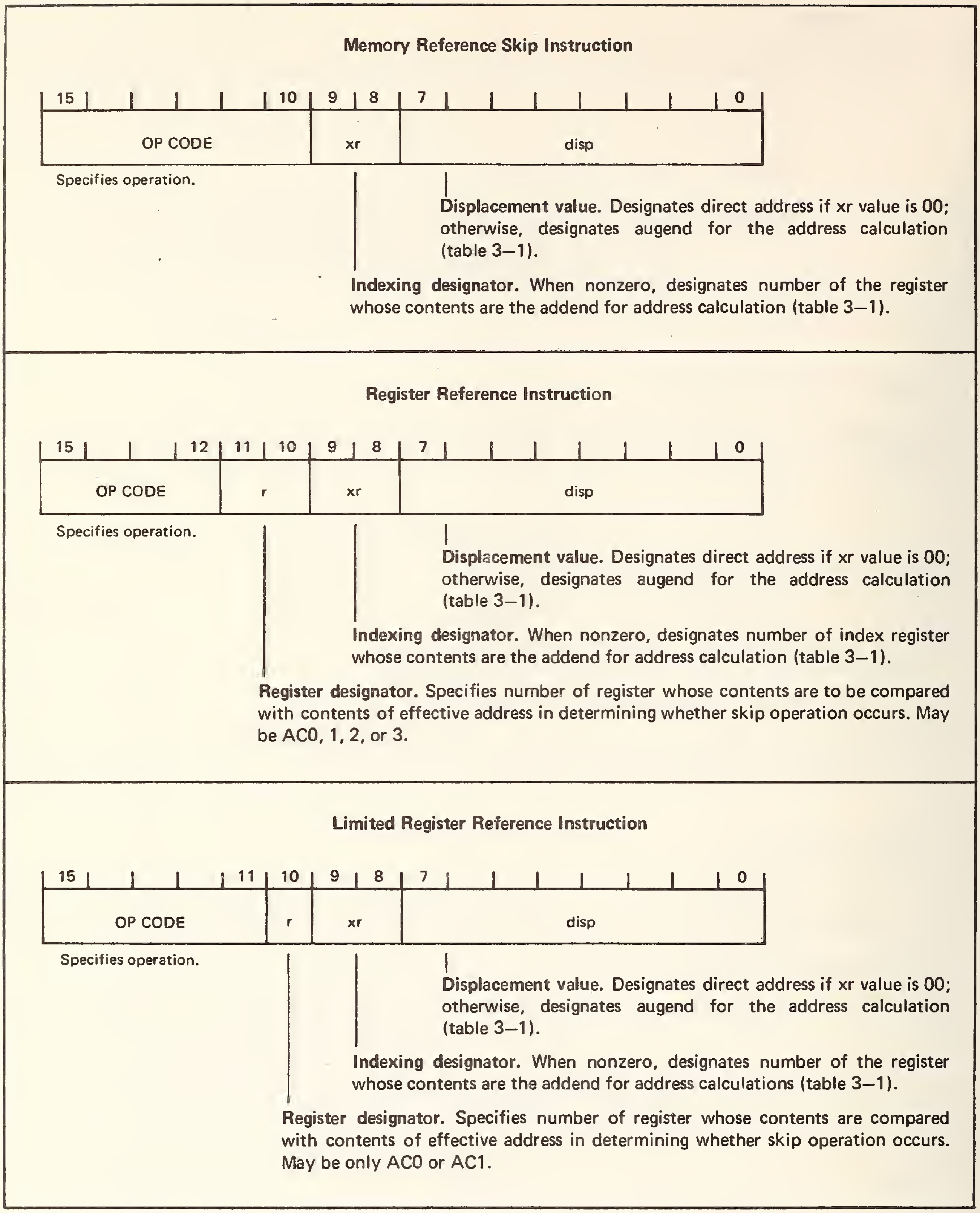
*Description:* The contents of ACr (where r is either 0 or 1) and the contents of the effective memory location EA are ORed inclusively, and the result is stored in ACr. The initial contents of ACr are lost, and the contents of EA are unaltered.

### 3.6.4 SKIP INSTRUCTIONS

Five instructions comprise the skip instructions, summarized in table 3–6. Three word formats are required and shown in figure 3–6.

*Table 3–6. Skip Instructions*

| Instruction | Operation Code | Operation | Assembler Format |
|---|---|---|---|
| **Memory References** | | | |
| INCREMENT AND SKIP IF ZERO | 011110 | (EA) ← (EA) + 1;<br>IF (EA) = 0, (PC) ← (PC) + 1 | ISZ disp(xr) |
| DECREMENT AND SKIP IF ZERO | 011111 | (EA) ← (EA) – 1;<br>IF (EA) = 0, (PC) ← (PC) + 1 | DSZ disp(xr) |
| **Register References** | | | |
| SKIP IF GREATER | 1110 | IF (ACr) > (EA), (PC) ← (PC) + 1 | SKG r, disp(xr) |
| SKIP IF NOT EQUAL | 1111 | IF (ACr) ≠ (EA), (PC) ← (PC) + 1 | SKNE r, disp(xr) |
| **Limited Register Reference** | | | |
| SKIP IF AND IS ZERO | 01110 | IF [(ACr) ∧ (EA)] = 0,<br>(PC) ← (PC) + 1 | SKAZ r, disp(xr) |


Memory Reference Skip Instruction
15
10
9
8
7
0
OP CODE
xr
disp
Register Reference Instruction
15
12
11
10
9
8
7
0
OP CODE
r
xr
disp
Limited Register Reference Instruction
15
11
10
9
8
7
0
OP CODE
r
xr
disp

Memory Reference Skip Instruction

Specifies operation.

**Displacement value.** Designates direct address if xr value is 00; otherwise, designates augend for the address calculation (table 3–1).

**Indexing designator.** When nonzero, designates number of the register whose contents are the addend for address calculation (table 3–1).

Register Reference Instruction

Specifies operation.

**Displacement value.** Designates direct address if xr value is 00; otherwise, designates augend for the address calculation (table 3–1).

**Indexing designator.** When nonzero, designates number of index register whose contents are the addend for address calculation (table 3–1).

**Register designator.** Specifies number of register whose contents are to be compared with contents of effective address in determining whether skip operation occurs. May be AC0, 1, 2, or 3.

Limited Register Reference Instruction

Specifies operation.

**Displacement value.** Designates direct address if xr value is 00; otherwise, designates augend for the address calculation (table 3–1).

**Indexing designator.** When nonzero, designates number of the register whose contents are the addend for address calculations (table 3–1).

**Register designator.** Specifies number of register whose contents are compared with contents of effective address in determining whether skip operation occurs. May be only AC0 or AC1.

*Figure 3–6. Skip Instruction Formats*

### Increment and Skip If Zero (ISZ)

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 0' | xr | | disp | | | | | | | |

*Operation:* (EA) ← (EA) + 1; if (EA) = 0, (PC) ← (PC) + 1

*Description:* The contents of EA are incremented by 1. The new contents of EA are tested to determine whether they equal zero. If the new contents of EA equal zero, the contents of PC are incremented by 1, thus skipping the memory location designated by the initial contents of PC. The contents of EA are unaltered.

### Decrement and Skip If Zero (DSZ)

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | xr | | disp | | | | | | | |

*Operation:* (EA) ← (EA) – 1; if (EA) = 0, (PC) ← (PC) + 1

*Description:* The contents of EA are decremented by 1. The new contents of EA are tested to determine whether they equal zero. If the new contents of EA equal zero, the contents of PC are incremented by 1, thus skipping the memory location designated by the initial contents of PC.

### Skip If Greater (SKG)

| 15 | | | 12 | 11 | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | r | | xr | | disp | | | | | | | |

*Operation:* If (ACr) > (EA), (PC) ← (PC) + 1

*Description:* The contents of ACr (when r is AC0, 1, 2, or 3) and the contents of the effective memory location EA are compared on an algebraic basis with due regard to the signs of the two operands. If the contents of ACr are greater than the contents of EA, the contents of PC are incremented by 1, thus skipping the memory location designated by the initial contents of PC. The initial contents of PC are lost. The contents of ACr and EA are unaltered.

**Skip If Not Equal (SKNE)**

| 15 | | | 12 | 11 | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | r | | xr | | disp | | | | | | | |

*Operation:* If ACr ≠ (EA), (PC) ← (PC) + 1

*Description:* The contents of ACr (where ACr is AC0, 1, 2, or 3) and the contents of the effective memory location EA are compared. If the contents of ACr and the effective memory location EA are not equal, the contents of PC are incremented, thus skipping the memory location designated by the initial contents of PC. The initial contents of PC are lost. The contents of ACr and EA are unaltered.

**Skip If AND Is Zero (SKAZ)**

| 15 | | | | 11 | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | r | xr | | disp | | | | | | | |

*Operation:* If [(ACr) ∧ (EA)] = 0, (PC) ← (PC) + 1

*Description:* The contents of ACr (where r is either 0 or 1) and the contents of the effective memory location EA are ANDed. If the result equals zero, the contents of PC are incremented by 1, thus skipping the instruction designated by the initial contents of PC. The initial contents of PC are lost. The contents of ACr and EA are unaltered.

### 3.6.5 TRANSFER-OF-CONTROL INSTRUCTIONS

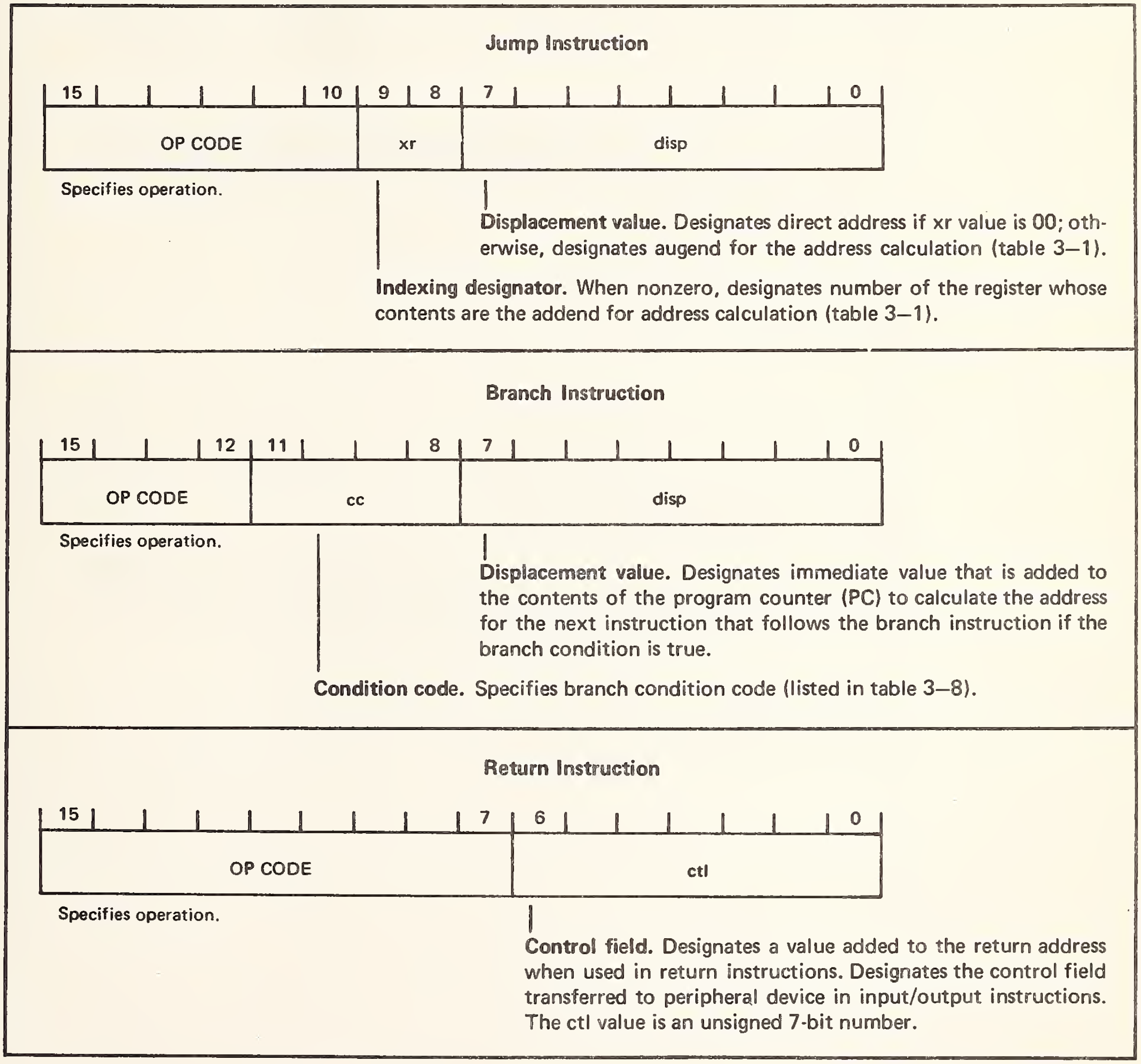
There are seven instructions in this group, summarized in table 3–7. Three word formats are required and shown in figure 3–7.

*Table 3–7. Transfer-of-Control Instructions*

| Instruction | Operation Code | Operation | Assembler Format |
|---|---|---|---|
| Jumps | | | |
| JUMP | 001000 | (PC) ← EA | JMP disp(xr) |
| JUMP INDIRECT | 001001 | (PC) ← (EA) | JMP @disp(xr) |
| JUMP TO SUBROUTINE | 001010 | (STK) ← (PC); (PC) ← EA | JSR disp(xr) |
| JUMP TO SUBROUTINE INDIRECT | 001011 | (STK) ← (PC); (PC) ← (EA) | JSR @disp(xr) |
| Branch | | | |
| BRANCH-ON CONDITION | 0001 | IF CC IS TRUE<br>(PC) ← (PC) + disp | BOC cc, disp |

*Table 3–7. Transfer-of-Control Instructions (Continued)*

| Instruction | Operation Code | Operation | Assembler Format |
|---|---|---|---|
| **Returns** | | | |
| RETURN FROM INTERRUPT | 000000010 | (PC) ← (STK) + ctl;<br>INTEN FLAG SET | RTI ctl |
| RETURN FROM SUBROUTINE | 000000100 | (PC) ← (STK) + ctl | RTS ctl |
| JUMP TO SUBROUTINE IMPLIED | 000000111 | (STK) ← (PC);<br>(PC) ← $FF80_{16}$ + ctl | JSRI ctl |



Specifies operation.

**Displacement value.** Designates direct address if xr value is 00; otherwise, designates augend for the address calculation (table 3–1).

**Indexing designator.** When nonzero, designates number of the register whose contents are the addend for address calculation (table 3–1).

Specifies operation.

**Displacement value.** Designates immediate value that is added to the contents of the program counter (PC) to calculate the address for the next instruction that follows the branch instruction if the branch condition is true.

**Condition code.** Specifies branch condition code (listed in table 3–8).

Specifies operation.

**Control field.** Designates a value added to the return address when used in return instructions. Designates the control field transferred to peripheral device in input/output instructions. The ctl value is an unsigned 7-bit number.

*Figure 3–7. Transfer-of-Control Instruction Formats*

### Jump(JMP)

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | xr | | disp | | | | | | | |

*Operation:* (PC) ← EA

*Description:* The effective address EA replaces the contents of PC. The next instruction is fetched from the location designated by the new contents of PC.

### Jump (JMP) Indirect

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | xr | | disp | | | | | | | |

*Operation:* (PC) ← (EA)

*Description:* The contents of the effective address (EA) replaces the contents of PC. The next instruction is fetched from the location designated by the new contents of PC.

### Jump to Subroutine (JSR)

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | xr | | disp | | | | | | | |

*Operation:* (STK) ← (PC), (PC) ← (EA)

*Description:* The contents of PC are stored in the top of the stack. The effective address EA replaces the contents of PC. The next instruction is fetched from the location designated by the new contents of PC.

### Jump to Subroutine (JSR) Indirect

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 1 | xr | | disp | | | | | | | |

*Operation:* (STK) ← (PC), (PC) ← (EA)

*Description:* The contents of PC are stored in the top of the stack. The contents of the effective address (EA) replace the contents of PC. The next instruction is fetched from the location designated by the new contents of PC.

### Branch-On Condition (BOC)

| 15 | | | 12 | 11 | | | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | cc | | | | disp | | | | | | | |

*Operation:* (PC) ← (PC) + disp (sign extended from bit 7 through bit 15)

*Description:* There are 16 possible condition codes (cc). These are listed in table 3–8. If the condition for branching designated by cc is true, the value of disp (sign extended from bit 7 through bit 15) is added to the contents of PC, and the sum is stored in PC. The initial contents of PC are lost. Program control is transferred to the location specified by the new contents of PC.

**NOTE**

(1) PC is always incremented by 1 immediately following the fetching of an instruction, so the contents of PC during execution of an instruction is 1 greater than the address of that instruction. This must be considered during execution of the BOC instruction: for example, if the address of the BOC instruction is 100, then 101 is added to the value of disp (sign extended).

(2) The disp field is a signed 8-bit number, whose sign is extended from bit 7 through bit 15 to form a 16-bit number (including sign). Thus, the range of addressing with a BOC instruction is -127 to +128 relative to the address of the current instruction.

*Table 3–8. Branch-On Condition Codes*

| Condition Code | Condition Tested | Remarks |
|---|---|---|
| 0000 | Interrupt Line = 1 | Interrupt need not be tested by macroprogram |
| 0001 | (AC0) = 0 | |
| 0010 | (AC0) ⩾ 0 | |
| 0011 | Bit 0 of AC0 = 1 | |
| 0100 | Bit 1 of AC0 = 1 | |
| 0101 | (AC0) ≠ 0 | |
| 0110 | CONTROL PANEL INTERRUPT LINE = 1 | |
| 0111 | CONTROL PANEL START = 1 | |
| 1000 | STACK FULL LINE = 1 | |
| 1001 | INTERRUPT ENABLE = 1 | |
| 1010 | CARRY/OVERFLOW = 1 | Carry if SEL = 0; overflow if SEL = 1 |
| 1011 | (AC0) ⩽ 0 | |
| 1100 | User | Available for general-purpose use |
| 1101 | User | |
| 1110 | User | |
| 1111 | User | |

**NOTE**

For both the following instructions (RTI and RTS), the ctl value is an unsigned 7-bit number.

**Return from Interrupt (RTI)**

| 15 | | | | | | | | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | ctl | | | | | | |

*Operation:* Set INT EN (interrupt enable flag)
(PC) ← (STK) + ctl

*Description:* The interrupt enable flag (INT EN) is set. The contents of PC are replaced by the sum of ctl and the contents of STK. Program control is transferred to the location specified by the new contents of PC. (RTI is used primarily to exit from an interrupt routine.)

**Return from Subroutine (RTS)**

| 15 | | | | | | | | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | ctl | | | | | | |

*Operation:* (PC) ← (STK) + ctl

*Description:* The contents of PC are replaced by the sum of ctl and the contents of STK. Program control is transferred to the location specified by the new contents of PC. (RTS is used primarily to return from subroutines entered by JSR.)

**Jump to Subroutine Implied (JSRI)**

| 15 | | | | | | | | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | ctl | | | | | | |

*Operation:* (STK) ← (PC), (PC) ← $FF80_{16}$ + ctl

*Description:* The contents of PC are pushed onto the stack. The contents of PC are replaced by the address implied by the sum of the ctl value and the number $FF80_{16}$. This enables a subroutine jump to memory locations $FF80_{16}$ through $FFFF_{16}$ ($0 \leqslant ctl \leqslant 7F_{16}$).

### 3.6.6 SHIFT INSTRUCTIONS

Four instructions comprise this group. All four instructions may be used with the Link (L) bit by setting the SEL flag. This is accomplished with a Set Flag (SFLG) instruction before executing the shift or rotate instruction. Examples of shifting with and without SEL set are given in diagram form for each instruction in the descriptions that follow. Note that the SEL flag also affects the BOC instructions as indicated in table 3–8.

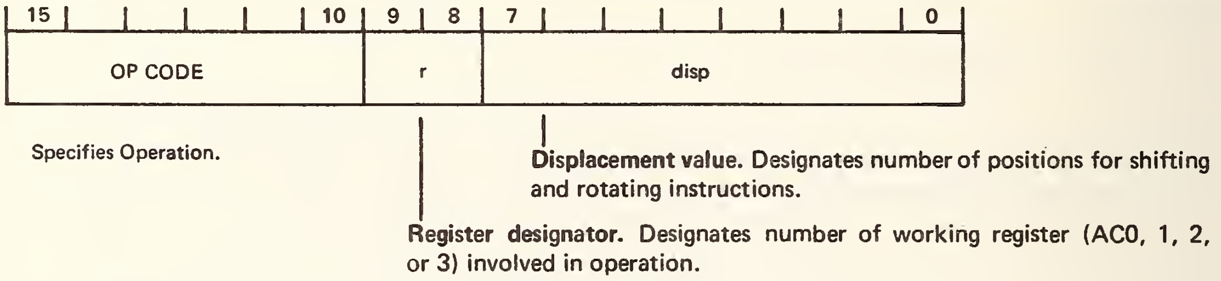
The shift instructions are summarized in table 3–9, and the word format is shown in figure 3–8.

All shift and rotate operations may be carried out with any of the four general-purpose accumulators, AC0, 1, 2, or 3.

*Table 3–9. Shift Instructions*

| Instruction | Operation Code | Operation | | Assembler Format |
|---|---|---|---|---|
| | | SEL = 0 | SEL = 1 | |
| ROTATE LEFT<br>(disp > 0) | 010110 | $(ACr_0) \leftarrow (ACr_{15})$,<br>$(ACr_n) \leftarrow (ACr_{n-1})$ | $(ACr_0) \leftarrow (L)$,<br>$(L) \leftarrow (ACr_{15})$,<br>$(ACr_n) \leftarrow (ACr_{n-1})$ | ROL r, m |
| ROTATE RIGHT<br>(disp < 0) | 010110 | $(ACr_{15}) \leftarrow (ACr_0)$,<br>$(ACr_n) \leftarrow (ACr_{n+1})$ | $(ACr_{15}) \leftarrow (L)$,<br>$(L) \leftarrow (ACr_0)$,<br>$(ACr_n) \leftarrow (ACr_{n+1})$ | ROR r, m |
| SHIFT LEFT<br>(disp > 0) | 010111 | $(ACr_n) \leftarrow (ACr_{n-1})$,<br>$(ACr_0) \leftarrow 0$ | $(L) \leftarrow (ACr_{15})$,<br>$(ACr_n) \leftarrow (ACr_{n-1})$,<br>$(ACr_0) \leftarrow 0$ | SHL r, m |
| SHIFT RIGHT<br>(disp < 0) | 010111 | $(ACr_{15}) \leftarrow 0$,<br>$(ACr_n) \leftarrow (ACr_{n+1})$ | $(ACr_{15}) \leftarrow (L)$,<br>$(L) \leftarrow 0$,<br>$(ACr_n) \leftarrow (ACr_{n+1})$ | SHR r, m |

NOTE: For all shift and rotate instructions, "m" denotes the number of positions to be shifted or rotated, and is equal to the absolute value of disp. See example 3 in appendix B.



Specifies Operation.

Displacement value. Designates number of positions for shifting and rotating instructions.

Register designator. Designates number of working register (AC0, 1, 2, or 3) involved in operation.

*Figure 3–8. Shift Instruction Format*

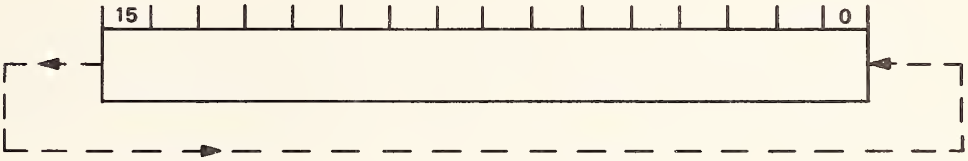
**Rotate Left (ROL)** *(for disp > 0)*

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0, | r | | disp | | | | | | | |

**SEL = 0**

*Operation:* $(ACr_0) \leftarrow (ACr_{15}), \quad (ACr_n) \leftarrow (ACr_{n-1})$

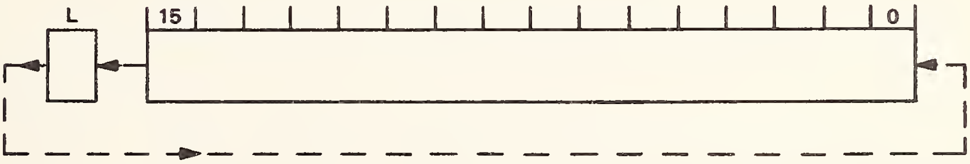
*Description:* The contents of ACr are shifted around to the left disp times. $(ACr_{15})$ replaces $(ACr_0)$ for each shift.



**SEL = 1**

*Operation:* $(ACr_0) \leftarrow (L), \quad (L) \leftarrow (ACr_{15}), \quad (ACr_n) \leftarrow (ACr_{n-1})$

*Description:* The contents of ACr are shifted around to the left disp times. (L) replaces $(ACr_0)$, and $(ACr_{15})$ replaces (L) for each shift.



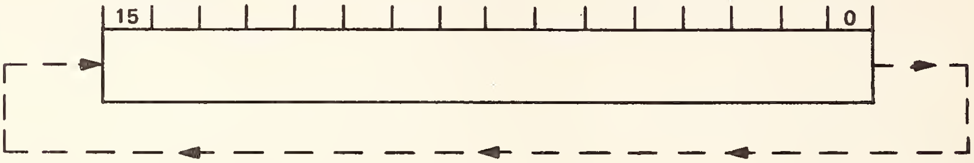
Rotate Right (ROR) *(for disp < 0)*

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | r | | disp | | | | | | | |

**SEL = 0**

*Operation:* $(ACr_{15}) \leftarrow (ACr_0)$, $(ACr_n) \leftarrow (ACr_{n+1})$

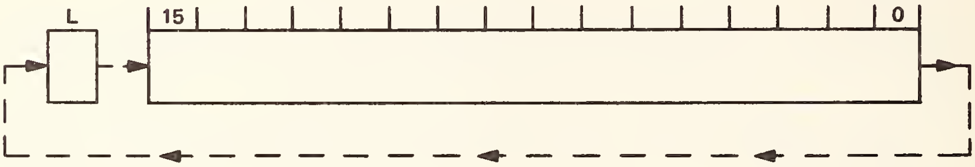
*Description:* The contents of ACr are shifted around to the right disp times. $(ACr_0)$ replaces $(ACr_{15})$ for each shift.



**SEL = 1**

*Operation:* $(ACr_{15}) \leftarrow (L)$, $(L) \leftarrow (ACr_0)$, $(ACr_n) \leftarrow (ACr_{n+1})$

*Description:* The contents of ACr are shifted around to the right disp times. (L) replaces $(ACr_{15})$, and $(ACr_0)$ replaces (L) for each shift.



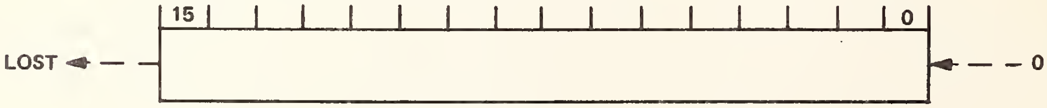
Shift Left (SHL) *(for disp > 0)*

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | r | | disp | | | | | | | |

**SEL = 0**

*Operation:* $(ACr_n) \leftarrow (ACr_{n-1})$, $(ACr_0) \leftarrow 0$

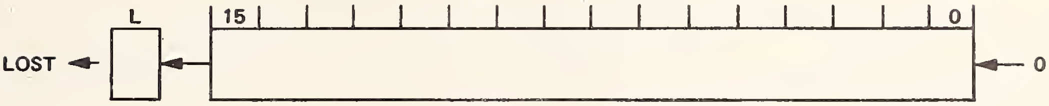
*Description:* The contents of ACr are shifted to the left disp times. $(ACr_{15})$ is lost, and zero replaces $(ACr_0)$ for each shift.



**SEL = 1**

*Operation:* (L) ← ($ACr_{15}$), ($ACr_n$) ← ($ACr_{n-1}$), ($ACr_0$) ← 0

*Description:* The contents of ACr are shifted to the left disp times. (L) is lost. ($ACr_{15}$) replaces (L), and zero replaces ($ACr_0$) for each shift.



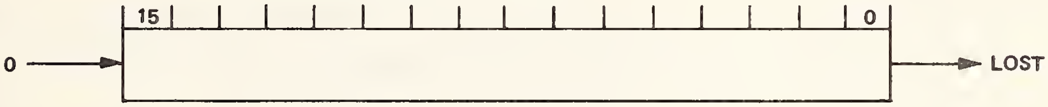
**Shift Right (SHR)** *(for disp < 0)*

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | r | | disp | | | | | | | |

**SEL = 0**

*Operation:* ($ACr_{15}$) ← 0, ($ACr_n$) ← ($ACr_{n+1}$)

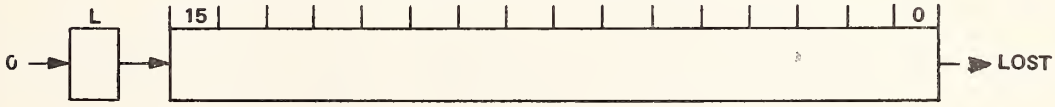
*Description:* The contents of ACr are shifted to the right disp times. ($ACr_0$) is lost, and zero replaces ($ACr_{15}$) for each shift.



**SEL = 1**

*Operation:* ($ACr_{15}$) ← (L), (L) ← 0, ($ACr_n$) ← ($ACr_{n+1}$)

*Description:* The contents of ACr are shifted to the right disp times. ($ACr_0$) is lost, (L) replaces $ACr_{15}$, and zero replaces (L) for each shift.



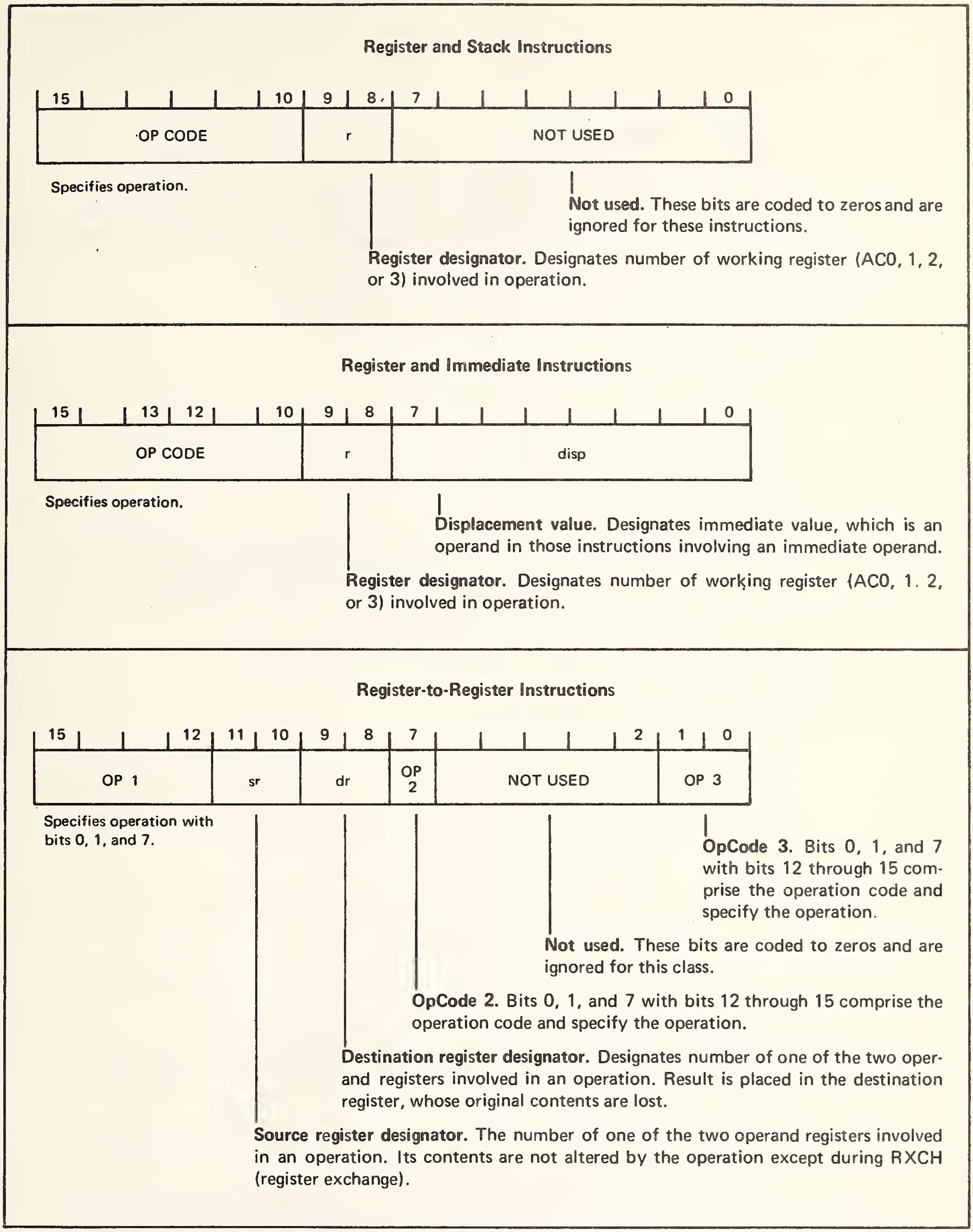
### 3.6.7 REGISTER INSTRUCTIONS

There are eleven instructions in this group, summarized in table 3–10. Three word formats are required and are shown in figure 3–9.

*Table 3–10. Register Instructions*

| Instruction | Operation Code | Operation | Assembler Format |
|---|---|---|---|
| Register and Stack | | | |
| PUSH ONTO STACK | 010000 | (STK) ← (ACr) | PUSH r |
| PULL FROM STACK | 010001 | (ACr) ← (STK) | PULL r |
| EXCHANGE REGISTER AND STACK | 010101 | (STK) ← (ACr), (ACr) ← (STK) | XCHRS r |
| Register and Immediate | | | |
| LOAD IMMEDIATE | 010011 | (ACr) ← disp (sign extended) | LI r, disp |
| ADD IMMEDIATE, SKIP IF ZERO | 010010 | (ACr) ← (ACr) + disp (sign extended), OV,CY; if (ACr) = 0, (PC) ← (PC) + 1 | AISZ r, disp |
| COMPLEMENT AND ADD IMMEDIATE | 010100 | (ACr) ← ~ (ACr) + disp (sign extended) | CAI r, disp |

| Instruction | Operation Code | | | Operation | Assembler Format |
|---|---|---|---|---|---|
| | OP1 | OP2 | OP3 | | |
| Register to Register | | | | | |
| REGISTER ADD | 0011 | 0 | 00 | (ACdr) ← (ACsr) + (ACdr), OV, CY | RADD sr, dr |
| REGISTER EXCHANGE | 0011 | 1 | 00 | (ACsr) ← (ACdr), (ACdr) ← (ACsr) | RXCH sr, dr |
| REGISTER COPY | 0011 | 1 | 01 | (ACdr) ← (ACsr) | RCPY sr, dr |
| REGISTER EXCLUSIVE-OR | 0011 | 1 | 10 | (ACdr) ← (ACsr) ⊽ (ACdr) | RXOR sr, dr |
| REGISTER AND | 0011 | 1 | 11 | (ACdr) ← (ACsr) ∧ (ACdr) | RAND sr, dr |


Register and Stack Instructions
15
10
9
8
7
0
OP CODE
r
NOT USED
Register and Immediate Instructions
15
13
12
10
9
8
7
0
OP CODE
r
disp
Register-to-Register Instructions
15
12
11
10
9
8
7
2
1
0
OP 1
sr
dr
OP 2
NOT USED
OP 3

**Register and Stack Instructions**

Specifies operation.

**Not used.** These bits are coded to zeros and are ignored for these instructions.

**Register designator.** Designates number of working register (AC0, 1, 2, or 3) involved in operation.

**Register and Immediate Instructions**

Specifies operation.

**Displacement value.** Designates immediate value, which is an operand in those instructions involving an immediate operand.

**Register designator.** Designates number of working register (AC0, 1. 2, or 3) involved in operation.

**Register-to-Register Instructions**

Specifies operation with bits 0, 1, and 7.

**OpCode 3.** Bits 0, 1, and 7 with bits 12 through 15 comprise the operation code and specify the operation.

**Not used.** These bits are coded to zeros and are ignored for this class.

**OpCode 2.** Bits 0, 1, and 7 with bits 12 through 15 comprise the operation code and specify the operation.

**Destination register designator.** Designates number of one of the two operand registers involved in an operation. Result is placed in the destination register, whose original contents are lost.

**Source register designator.** The number of one of the two operand registers involved in an operation. Its contents are not altered by the operation except during RXCH (register exchange).

*Figure 3–9. Register Instruction Formats*

### Push onto Stack (PUSH)

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | r | | NOT USED | | | | | | | |

*Operation:* (STK) ← (ACr)

*Description:* The stack is pushed by the contents of the register (AC0, 1, 2, or 3) designated by r. Thus, the top of the stack then holds the contents of ACr, and the contents of all other levels in the stack are moved down one level. If the stack is full before the push occurs, the contents of the lowest level are lost. The initial contents of ACr are unaltered.

### Pull from Stack (PULL)

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | r | | NOT USED | | | | | | | |

*Operation:* (ACr) ← (STK)

*Description:* The stack is pulled. The contents from the top of the stack replace the contents of register number r (AC0, 1, 2, or 3). The initial contents of ACr are lost. The contents of each level of the stack moves up one level. Zeros enter the bottom of the stack.

### Exchange Register and Stack (XCHRS)

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | r | | NOT USED | | | | | | | |

*Operation:* (STK) ← (ACr), (ACr) ← (STK)

*Description:* The contents of the top of the stack STK and the register designated by r (AC0, 1, 2, or 3) are exchanged.

### Load Immediate (LI)

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | r | | disp | | | | | | | |

*Operation:* (ACr) ← disp (sign extended)

*Description:* The value of disp with sign bit 7 extended through bit 15 replaces the contents of ACr (AC0, 1, 2, or 3). The initial contents of ACr are lost. The immediate operand range is –128 to +127.

### Add Immediate, Skip If Zero (AISZ)

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | r | | disp | | | | | | | |

*Operation:* (ACr) ← (ACr) + disp (sign extended), OV, CY
If new (ACr) = 0, (PC) ← (PC) + 1

*Description:* The contents of register ACr are replaced by the sum of the contents of ACr and disp (sign bit 7 extended through bit 15). The initial contents of ACr are lost. The overflow and carry flags are set according to the result of the operation. If the new contents of ACr equal zero, the contents of PC are incremented by 1, thus skipping the next memory location. The immediate operand range is -128 to +127.

### Complement and Add Immediate (CAI)

| 15 | | | | | 10 | 9 | 8 | 7 | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | r | | disp | | | | | | | |

*Operation:* (ACr) ← ~ (ACr) + disp (sign extended)

*Description:* The contents of register ACr are complemented and then added to disp (sign bit extended through bit 15). The result is then stored in ACr. The initial contents of ACr are lost. The immediate operand range is –128 to +127. Note that the carry and overflow flags are not affected by this instruction.

### Register Add (RADD)

*Operation:* (ACdr) ← (ACsr) + (ACdr), OV, CY

*Description:* The contents of the destination register ACdr (AC0, 1, 2, or 3) are replaced by the sum of the contents of ACdr and the source register ACsr (AC0, 1, 2, or 3). The initial contents of ACdr are lost, and the contents of ACsr are unaltered. The overflow and carry flags are set according to the result of the operation.

### Register Exchange (RXCH)

| 15 | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | | | | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | sr | | dr, | | 1 | NOT USED | | | | | 0 | 0 |

*Operation:* (ACsr) ← (ACdr), (ACdr) ← (ACsr)

*Description:* The contents of ACsr (AC0, 1, 2, or 3) and ACdr (AC0, 1, 2, or 3) are exchanged.

### Register Copy (RCPY)

| 15 | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | | | | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | sr | | dr | | 1 | NOT USED | | | | | 0 | 1 |

*Operation:* (ACdr) ← (ACsr)

*Description:* The contents of the destination register ACdr (AC0, 1, 2, or 3) are replaced by the contents of the source register ACsr (AC0, 1, 2, or 3). The initial contents of ACdr are lost, and the initial contents of ACsr are unaltered.

### Register Exclusive Or (RXOR)

| 15 | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | | | | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | sr | | dr | | 1 | NOT USED | | | | | 1 | 0 |

*Operation:* (ACdr) ← (ACdr) ∇ (ACsr)

*Description:* The contents of the destination register ACdr (AC0, 1, 2, or 3) are replaced by the result of exclusively ORing the contents of ACdr with the contents of the source register ACsr (AC0, 1, 2, or 3). The initial contents of ACdr are lost, and the initial contents of ACsr are unaltered.

### Register And (RAND)

| 15 | | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | | | | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | sr | | dr | | 1 | NOT USED | | | | | 1 | 1 |

*Operation:* (ACdr) ← (ACdr) ∧ (ACsr)

*Description:* The contents of the destination register ACdr (AC0, 1, 2, or 3) are replaced by the result of ANDing the contents of ACdr with the contents of the source register ACsr (AC0, 1, 2, or 3). The initial contents of ACdr are lost, and the initial contents of ACsr are unaltered.

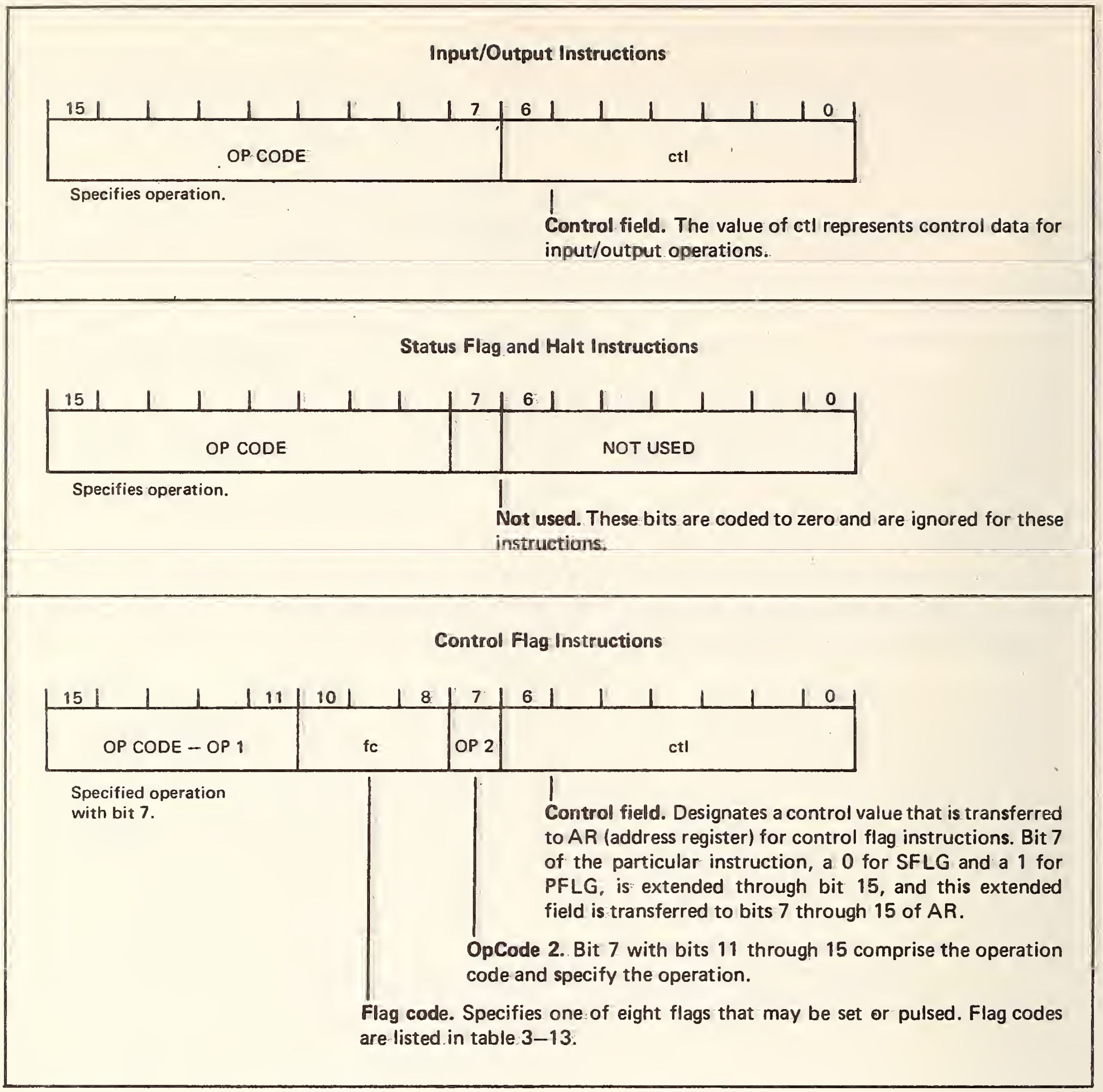
### 3.6.8 INPUT/OUTPUT, HALT, AND FLAG INSTRUCTIONS

Seven instructions comprise this group, summarized in table 3–11. Three word formats are required and are shown in figure 3–10.

*Table 3–11. Input/Output, Halt, and Flag Instructions*

| Instruction | Operation Code | Operation | Assembler Format |
|---|---|---|---|
| **Input/Output** | | | |
| REGISTER IN | 000001000 | (AR)←ctl + (AC3);<br>(AC0)←(IOREG) | RIN ctl |
| REGISTER OUT | 000001100 | (AR)←ctl + (AC3);<br>(IOREG)←(AC0) | ROUT ctl |
| **Halt** | | | |
| HALT | 000000000 | Processor halts. | HALT |
| **Status Flags** | | | |
| PUSH STATUS FLAGS ONTO STACK | 000000001 | (STK) ← (STATUS FLAGS) | PUSHF |
| PULL STATUS FLAGS FROM STACK | 000000101 | (STATUS FLAGS) ← (STK) | PULLF |

| Instruction | Operation Code | | Operation | Assembler Format |
|---|---|---|---|---|
| | OP1 | OP2 | | |
| **Control Flags** | | | | |
| SET FLAG | 00001 | 0 | fc set; (AR)←ctl | SFLG fc |
| PULSE FLAG | 00001 | 1 | fc pulsed; (AR)←ctl | PFLG fc |



**Input/Output Instructions**

Specifies operation.

**Control field.** The value of ctl represents control data for input/output operations.

**Status Flag and Halt Instructions**

Specifies operation.

**Not used.** These bits are coded to zero and are ignored for these instructions.

**Control Flag Instructions**

Specified operation with bit 7.

**Control field.** Designates a control value that is transferred to AR (address register) for control flag instructions. Bit 7 of the particular instruction, a 0 for SFLG and a 1 for PFLG, is extended through bit 15, and this extended field is transferred to bits 7 through 15 of AR.

**OpCode 2.** Bit 7 with bits 11 through 15 comprise the operation code and specify the operation.

**Flag code.** Specifies one of eight flags that may be set or pulsed. Flag codes are listed in table 3–13.

*Figure 3–10. Input/Output, Halt, and Flag Instruction Formats*

### Register In (RIN)

| 15 | | | | | | | | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ctl | | | | | | |

*Operation:* (AR) ← ctl + (AC3), (AC0) ← (IOREG)

*Description:* The contents of AR (address register) are replaced by the sum of ctl and the contents of AC3. The new contents of AR constitute the address of a peripheral device and a command, both of which are received by the addressed peripheral device. The peripheral device responds by transferring the contents of its input/output register (IOREG) to the processor AC0.

### Register Out (ROUT)

| 15 | | | | | | | | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | ctl | | | | | | |

*Operation:* (AR) ← ctl + (AC3), (IOREG) ← (AC0)

*Description:* The contents of AR (address register) are replaced by the sum of ctl and the contents of AC3. The new contents of AR constitute the address of a peripheral device and a command, both of which are received by the addressed peripheral device. The processor then transfers the contents of AC0 to IOREG in the peripheral device.

### Halt (HALT)

| 15 | | | | | | | | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | NOT USED | | | | | | |

*Description:* The processor halts and remains halted until the START input makes a transition from logic "1" to "0." A switch may be wired to this input such that a logic "1" is applied momentarily, and then released to a logic "0" level, thereby providing the necessary transition. The HALT flag is set by this instruction, and it remains set until the processor comes out of the halted condition.

### Push Status Flags onto Stack (PUSHF)

| 15 | | | | | | | | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | NOT USED | | | | | | |

*Operation:* (STK) ← (STATUS FLAGS)

*Description:* The contents of the top of the stack STK are replaced by the contents of the status flags. See figure 3–11 for the configuration of processor flags on the stack and table 3–12 for processor flag definitions. The previous contents of the top of the stack and lower levels are pushed down one level. The contents of the lowest level of the stack are lost.

### Pull Status Flags from Stack (PULLF)

| 15 | | | | | | | | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | NOT USED | | | | | | |

*Operation:* (STATUS FLAGS) ← (STK)

*Description:* The contents of the status flags are replaced by the contents of the top of the stack (STK). See figure 3–11 for the configuration of processor flags on the stack and table 3–12 for processor-flag definitions. The previous contents of lower levels of the stack are pulled up by one level with zeros replacing the contents of the lowest level.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Bit Positions |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| L | OV | CY | FLAG 12 | GF | GF | GF | GF | GF | GF | GF | GF | GF | GF | GF | FLAG 0 | Flags |

*Figure 3–11. Configuration of Status Flags*

**NOTE**

Status flags 0 and 12 are externally available.

*Table 3–12. Status Flags*

| Bit Position | Flag Name | Mnemonic | Significance |
|---|---|---|---|
| 15 | Link | L | Used for double-word shifts |
| 14 | Overflow | OV | Set if an arithmetic overflow occurs |
| 13 | Carry | CY | Set if a carry occurs (from most significant bit) during an arithmetic operation |
| 12 through 0 | General-Purpose Flags | GF | Use specified by programmer |

**Set Flag (SFLG)**

| 15 | | | | 11 | 10 | | 8 | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | fc | | | 0 | ctl | | | | | | |

*Operation:* FC set, (AR) ← ctl (bit 7 extended through bit 15; that is, bits 8 through 15 set to 0)

*Description:* The control flag designated by the flag code FC is set. The contents of the address register AR are replaced by the value of ctl. Flag codes are defined in table 3–13.

**Pulse Flag (PFLG)**

| 15 | | | | 11 | 10 | | 8 | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | fc | | | 1 | ctl | | | | | | |

*Operation:* FC pulsed, (AR) ← ctl (bit 7 extended through bit 15; that is, bits 8 through 15 set to 1)

*Description:* The control flag designated by the flag code FC is pulsed (note 2 below). The contents of the address register AR are replaced by the value of ctl. Flag codes are defined in table 3–13.

NOTE

(1) SFLG and PFLG refer to control flags external to the RALUs. These flags should not be confused with the RALU-internal flags, which are referenced by PUSHF and PULLF.

(2) Pulsing a control flag sets the flag at T2 and resets it at T6 during the same microcycle. The flag remains reset until again set or pulsed.

(3) The ctl value plus the extended bit 7 through bit 15 are transferred to the AR (address register). This word in the AR has no specified use and may be used as desired by the system programmer.

*Table 3–13. Control Flag Codes*

| FC | Flag Mnemonic | Significance |
|---|---|---|
| 000 | F8 | User Specified |
| 001 | INT EN | Interrupt Enable |
| 010 | SEL | Select |
| 011 | F11 | User Specified |
| 100 | F12 | User Specified |
| 101 | F13 | User Specified |
| 110 | F14 | User Specified |
| 111 | F15 | User Specified |

NOTE: The flag designated by the flag code (fc) is set or pulsed. Only control flags with addresses between 8 and 15 may be accessed with these instructions. (The flag address is 8, binary 1000, greater than the corresponding flag code, FC.) This is done because control flags with flag addresses 0 through 7 are used for various input/output operations controlled by the processor, and are not usable by the programmer. The SEL flag selects between CY and OV for output on the CYOV line; it also selects the LINK bit for inclusion in shift and rotate operations.

## 3.7 EXTENDED INSTRUCTION SET

An extended instruction set is available for the IMP-16C in the form of a second CROM. This set comprises 17 instructions divided into 5 categories:

- Double-word Arithmetic
- Load and Store Byte
- Bit Operations
- Interrupt Handling Operations
- Transfer of Control Operations

The instructions for each functional type are described as a group. For each instruction, the name of the instruction, its mnemonic, its word format, its operation in the form of an equation, and a succinct explanation of its operation are given. A tabulated summary of each type of instruction precedes the detailed descriptions.

### 3.7.1 DOUBLE-WORD MEMORY ADDRESSING

Six of the seventeen instructions use a double-word instruction format. These instructions use direct and indirect addressing exactly as described in 3.4, although the instruction format is different for these six memory-reference instructions from that described in 3.4. The six memory-reference instructions are as follows:

- MULTIPLY (MPY)
- DIVIDE (DIV)

- DOUBLE PRECISION ADD (DADD)
- DOUBLE PRECISION SUBTRACT (DSUB)
- LOAD BYTE (LDB)
- STORE BYTE (STB)

The modified format (shown in figure 3–12) is a double-word instruction format with a 16-bit displacement field. When using these instructions, all of memory is directly addressable, although all indexing modes may still be used. The addressing modes are unchanged from those indicated in table 3–1. It is important to note when considering PC relative addressing, that the PC contains the address of the displacement word of the instruction.

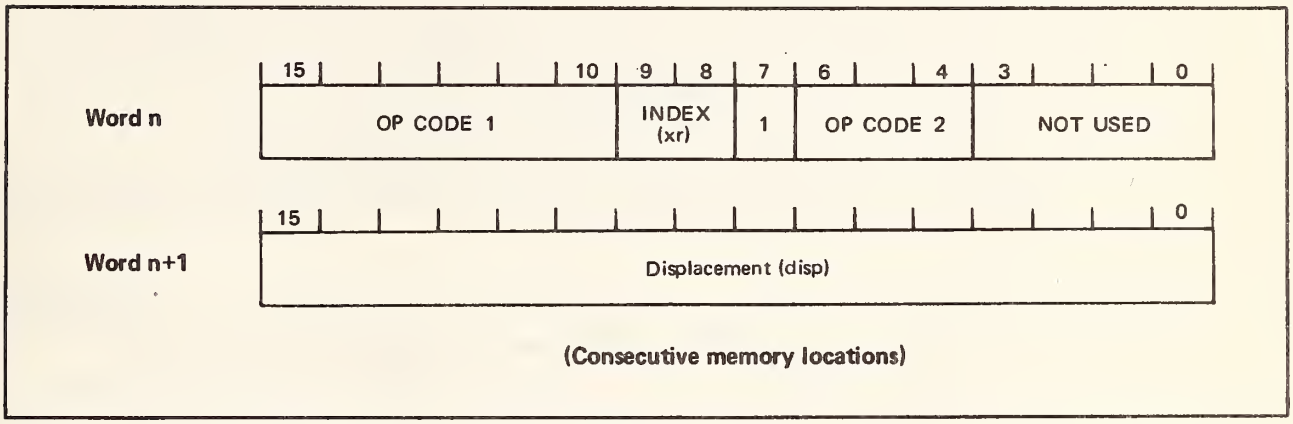


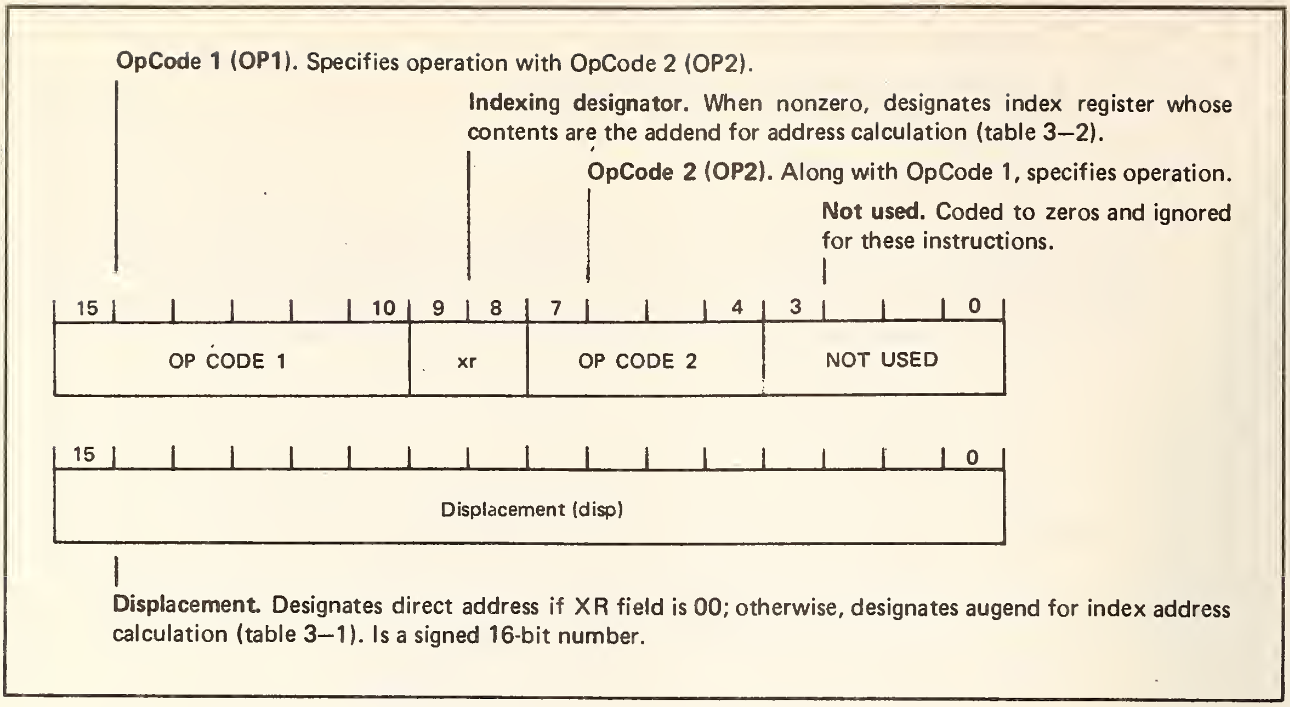
*Figure 3–12. Double-Word Memory Reference Instruction Format*

### 3.7.2 DOUBLE-WORD ARITHMETIC INSTRUCTIONS

There are four instructions in this group. These are summarized in table 3–14 and then individually described. The word format for the two double-precision instructions is shown in figure 3–13. The multiply and divide instructions have formats as shown in figure 3–13.

*Table 3–14. Double-Word Arithmetic Instructions*

| Instruction | Operation Code | | Operation | Assembler Format |
|---|---|---|---|---|
| | OP1 | OP2 | | |
| DOUBLE PRECISION ADD | 000001 | 1010 | (AC0), (AC1) ← (AC0), (AC1) + (EA), (EA+1); OV; CY; SEL ← 0 | DADD disp(xr) |
| DOUBLE PRECISION SUBTRACT | 000001 | 1011 | (AC0), (AC1) ← (AC0), (AC1) + ∿ (EA), ∿ (EA+1) + 1; OV; CY; SEL ← 0 | DSUB disp(xr) |
| MULTIPLY (MPY) ((EA) ⩾ 0) | 000001 | 1000 | (AC0), (AC1) ← (AC1)*(EA); SEL ← 0; L altered | MPY disp(xr) |
| DIVIDE (DIV) ((EA) > 0) | 000001 | 1001 | (AC0) ← INTEGER PART OF [(AC0), (AC1) ÷ (EA)];<br>(AC1) ← REMAINDER OF [(AC0), (AC1) ÷ (EA)];<br>OV; SEL ← 0; L altered | DIV disp(xr) |



**OpCode 1 (OP1).** Specifies operation with OpCode 2 (OP2).

**Indexing designator.** When nonzero, designates index register whose contents are the addend for address calculation (table 3–2).

**OpCode 2 (OP2).** Along with OpCode 1, specifies operation.

**Not used.** Coded to zeros and ignored for these instructions.

**Displacement.** Designates direct address if XR field is 00; otherwise, designates augend for index address calculation (table 3–1). Is a signed 16-bit number.

*Figure 3–13. Double-Precision (Double-Word) Arithmetic Instruction Format*

### Double-Precision Add (DADD)

| 15 | | | | | 10 | 9 | 8 | 7 | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | xr | | 1 | 0 | 1 | 0 | NOT USED | | | |

| 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| disp | | | | | | | | | | | | | | | |

*Operation:* (AC0), (AC1) ← (AC0), (AC1) + (EA), (EA+1); OV; CY; SEL ← 0

*Description:* The double-precision twos-complement value in AC0 (high order) and AC1 (low order) is added to the double-precision twos-complement value in EA (high order) and EA+1 (low order), and the result is stored in AC0 and AC1. The contents of EA and EA+1 are unchanged. The overflow or carry flag is set if an overflow or carry occurs, respectively; otherwise, they are cleared. The select flag is cleared.

### Double-Precision Subtract (DSUB)

| 15 | | | | | 10 | 9 | 8 | 7 | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | xr | | 1 | 0 | 1 | 1 | NOT USED | | | |

| 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| disp | | | | | | | | | | | | | | | |

*Operation:* (AC0), (AC1) ← (AC0), (AC1) + ∿ [(EA), (EA+1)] + 1; OV; CY; SEL ← 0

*Description:* The double-precision twos-complement value in EA (high order) and EA+1 (low order) is subtracted from the double-precision twos-complement value in AC0 (high order) and AC1 (low order). The contents of EA and EA+1 are unchanged. The overflow or carry flag is set if an overflow or carry occurs, respectively; otherwise, they are cleared. The select flag is cleared.

### Multiply (MPY)

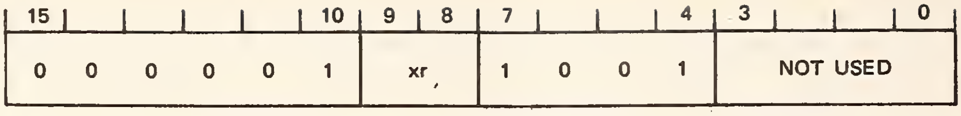
| 15 | | | | | 10 | 9 | 8 | 7 | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | xr | | 1 | 0 | 0 | 0 | NOT USED | | | |

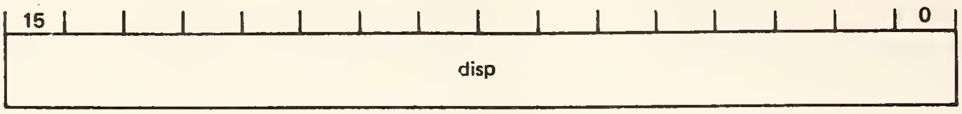
| 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| disp | | | | | | | | | | | | | | | |

*Operation:* (AC0), (AC1) ← (AC1) * (EA); SEL ← 0; L altered

*Description:* The unsigned integer in AC1 is multiplied by the positive integer in the effective memory location EA. The high-order part of the 32-bit result is stored in AC0 and the low-order part is stored in AC1. The previous contents of AC0 and AC1 are lost. The contents of EA are unaffected. The select flag is cleared. The link flag is left in an arbitrary state.

Divide (DIV)





*Operation:* (AC0), (AC1) ← (AC0), (AC1) ÷ (EA); OV; SEL ← 0; L altered

*Description:* The positive 32-bit integer in AC0 (high-order part) and AC1 (low-order part) is divided by the contents (a positive number) of the effective memory location EA. The integer quotient is placed in AC1 and the remainder in AC0. The overflow flag (OV) is set if either of the following occurs: (1) the high-order part of the dividend (initial contents of AC0) is greater than or equal to the divisor, or (2) the quotient is negative. The select flag is cleared. The link flag is left in an arbitrary state. The contents of EA are unchanged. Division by zero, an illegal operation, falls into case 1 above.

### 3.7.3 BYTE INSTRUCTIONS

There are two instructions in this category; both are double-word, memory reference instructions. They are summarized in table 3–15 and then individually described. Their word format is shown in figure 3–13.

*Table 3–15. Byte Instructions*

| Instruction | Operation Code | | Operation | Assembler Format |
|---|---|---|---|---|
| | OP1 | OP2 | | |
| LOAD BYTE | 000001 | 1100 | (Low-order byte of AC0) ← byte from (EA ÷ 2); SEL ← 0 | LDB disp(xr) |
| STORE BYTE | 000001 | 1101 | Byte of (EA ÷ 2) ← (low-order byte of AC0); SEL ← 0 | STB disp(xr) |

## LOAD BYTE (LDB)

| 15 | | | | | 10 | 9 | 8 | 7 | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | xr | | 1 | 1 | 0 | 0 | NOT USED | | | |

| 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| disp | | | | | | | | | | | | | | | |

*Operation:* Low-order byte of (AC0) ← byte from (EA ÷ 2); SEL ← 0

*Description:* The low-order byte of AC0 is loaded with a byte from (EA ÷ 2). If the low-order bit of EA is 1, the low-order byte is loaded; otherwise, the high-order byte is loaded. (Note: EA ÷ 2 is the effective address shifted right one position.) The high-order byte of AC0 is set equal to zero. The select flag is cleared.

## STORE BYTE (STB)

| 15 | | | | | 10 | 9 | 8 | 7 | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | xr | | 1 | 1 | 0 | 1 | NOT USED | | | |

| 15 | | | | | | | | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| disp | | | | | | | | | | | | | | | |

*Operation:* Byte of (EA ÷ 2) ← low-order byte from (AC0); SEL ← 0

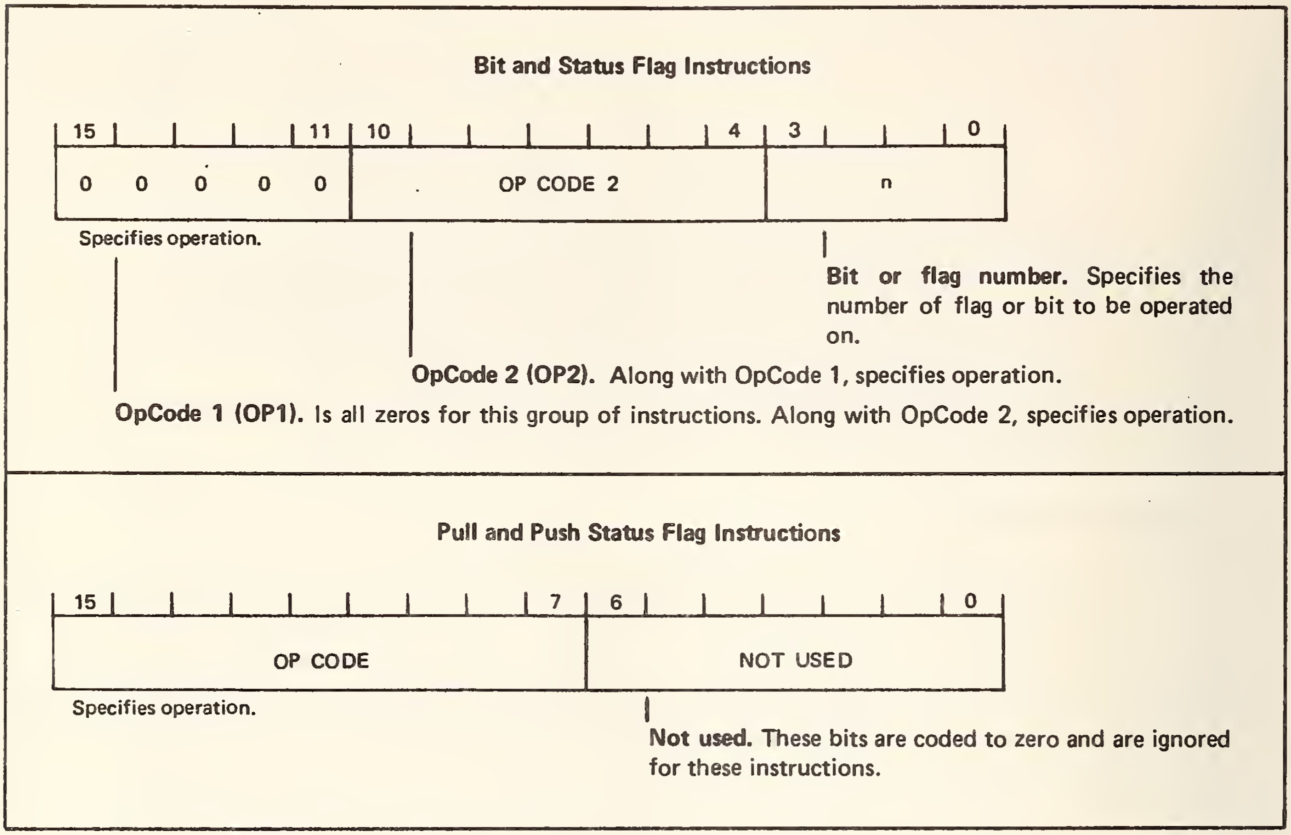
*Description:* The low-order byte of AC0 is stored into the byte of (EA ÷ 2) specified by the low-order bit of EA. If the low-order bit is 1, the low-order byte is specified; otherwise, the high-order byte is specified. (Note: EA ÷ 2 is the effective address shifted right one position.) The unspecified byte of EA ÷ 2 and the contents of AC0 are unaffected. The select flag is cleared.

### PROGRAMMING NOTE

The effective address is formed by adding the contents of the index register to the displacement (EA = XR + DISP). Byte addresses are formed by shifting this quantity right one bit position (EA ÷ 2 = [XR + DISP] /2 = XR/2 + DISP/2). Bit 0 of the EA specifies the left byte if equal to 1; the right bit if equal to zero.

### 3.7.4 BIT AND STATUS FLAG INSTRUCTIONS

There are seven single-word instructions in this group. They are summarized in table 3–16 and then described individually. Their word formats are shown in figure 3–14.


Bit and Status Flag Instructions
15
11
10
4
3
0
0 0 0 0 0
OP CODE 2
n
Pull and Push Status Flag Instructions
15
7
6
0
OP CODE
NOT USED

Specifies operation.

**Bit or flag number.** Specifies the number of flag or bit to be operated on.

**OpCode 2 (OP2).** Along with OpCode 1, specifies operation.

**OpCode 1 (OP1).** Is all zeros for this group of instructions. Along with OpCode 2, specifies operation.

Specifies operation.

**Not used.** These bits are coded to zero and are ignored for these instructions.

*Figure 3–14. Bit and Status Flag Instruction Formats*

*Table 3–16. Bit and Status Flag Single-Word Instructions*

| Instruction | OpCode 2 | Operation | Assembler Format |
|---|---|---|---|
| SET STATUS FLAG | 1110000 | Status Flag n ← 1; SEL ← 0 | SETST n |
| CLEAR STATUS FLAG | 1110001 | Status Flag n ← 0; SEL ← 0 | CLRST n |
| SET BIT | 1110010 | $AC0_n$ ← 1; SEL ← 0 | SETBIT n |
| CLEAR BIT | 1110011 | $AC0_n$ ← 0; SEL ← 0 | CLRBIT n |
| COMPLEMENT BIT | 1110110 | $AC0_n \leftarrow \sim AC0_n$; SEL ← 0 | CMPBIT n |
| SKIP IF STATUS FLAG TRUE | 1110100 | IF Status Flag n = 1, then (PC) ← (PC) + 1; SEL ← 0 | SKSTF n |
| SKIP IF BIT TRUE | 1110101 | IF $AC0_n$ = 1, then (PC) ← (PC) + 1; SEL ← 0 | SKBIT n |

## SET STATUS FLAG (SETST)

| 15 | | | | 11 | 10 | | | | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | n | | | |

*Operation:* Status Flag n ← 1; SEL ← 0

*Description:* Bit n of the status flag register is set true. All other bits are unaffected. The select flag is cleared. $(0 \leqslant n \leqslant 15)$

## CLEAR STATUS FLAG (CLRST)

| 15 | | | | 11 | 10 | | | | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | n | | | |

*Operation:* Status Flag n ← 0; SEL ← 0

*Description:* Bit n of the status flag register is cleared. All other bits are unaffected. The select flag is cleared. $(0 \leqslant n \leqslant 15)$

### SET BIT (SETBIT)

| 15 | | | | 11 | 10 | | | | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | n | | | |

*Operation:* $ACO_n \leftarrow 1$; SEL ← 0

*Description:* Bit n of AC0 is set true. All other bits are unaffected. The select flag is cleared. $(0 \leqslant n \leqslant 15)$

### CLEAR BIT (CLRBIT)

| 15 | | | | 11 | 10 | | | | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | n | | | |

*Operation:* $ACO_n \leftarrow 0$; SEL ← 0

*Description:* Bit n of AC0 is cleared. All other bits are unaffected. The select flag is cleared. $(0 \leqslant n \leqslant 15)$

### COMPLEMENT BIT (CMPBIT)

| 15 | | | | 11 | 10 | | | | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | n | | | |

*Operation:* $ACO_n \leftarrow \sim ACO_n$; SEL ← 0

*Description:* Bit n of AC0 is complemented. All other bits are unaffected. The select flag is cleared. $(0 \leqslant n \leqslant 15)$

### SKIP IF STATUS FLAG TRUE (SKSTF)

| 15 | | | | 11 | 10 | | | | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | n | | | |

*Operation:* IF Status Flag n = 1, (PC) ← (PC) + 1; SEL ← 0

*Description:* If Status Flag n is true, the next memory location is skipped. The contents of the status flags are unaffected. The select flag is cleared.

### SKIP IF BIT TRUE (SKBIT)

| 15 | | | | 11 | 10 | | | | | | 4 | 3 … 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | n |

*Operation:* IF $AC0_n = 1$, (PC) ← (PC) + 1; SEL ← 0

*Description:* If bit n of AC0 is true, the next memory location in sequence is skipped. The contents of AC0 are unaffected. The select flag is cleared.

**PROGRAMMING NOTE**

Caution should be taken when coding a skip instruction to prevent the skip condition from jumping into the displacement field of a double-word instruction.

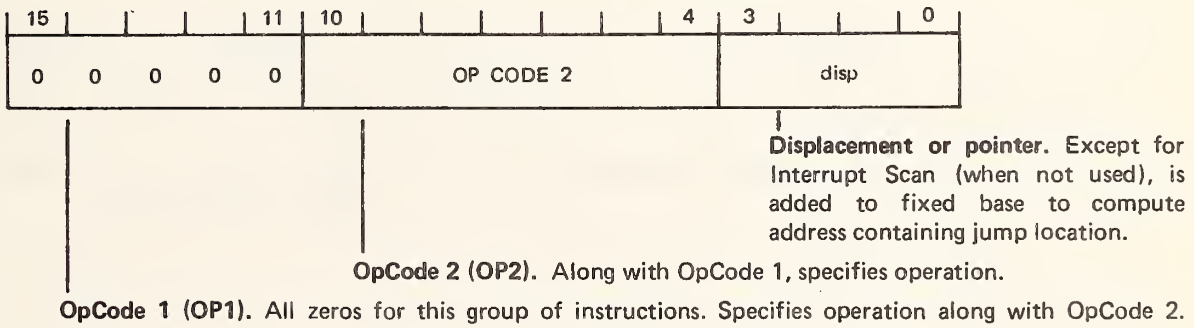
## 3.7.5 INTERRUPT HANDLING INSTRUCTIONS

There are two instructions in this group, listed in table 3–17. Their word format is shown in figure 3–15, and then described in succeeding paragraphs.

*Table 3–17. Interrupt Handling Instructions*

| Instruction | OpCode 2 | Operation | Assembler Format |
|---|---|---|---|
| INTERRUPT SCAN | 1010001 | If (AC1) = 0, SEL ← 0;<br>If (AC1) ≠ 0, SEL ← 0;<br>AC1 ← [shift AC1 right 1] until 1 shifted out<br>(AC2) ← (AC2) + number of shifts; (PC) ← (PC) + 1 | ISCAN |
| JUMP INDIRECT TO LEVEL 0 INTERRUPT | 1010010 | (STK) ← (PC); (PC) ← ([$120_{16}$ + disp])<br>INTEN ← 0 | JINT disp |

**ISCAN, JINT, and JMPP Instructions**



**Displacement or pointer.** Except for Interrupt Scan (when not used), is added to fixed base to compute address containing jump location.

**OpCode 2 (OP2).** Along with OpCode 1, specifies operation.

**OpCode 1 (OP1).** All zeros for this group of instructions. Specifies operation along with OpCode 2.

*Figure 3–15. Interrupt and Word Jump Formats*

INTERRUPT SCAN (ISCAN)

| 15 | | | | 11 | 10 | | | | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | NOT USED | | | |

Operation: If AC1 = 0, SEL ← 0;

```
Else,    SEL ← 0;
         (AC1) ← (shift AC1 right 1) until 1 shifted out;
         (AC2) ← (AC2) + number of shifts;
         (PC) ← (PC) + 1
```

*Description:* If AC1 = 0, the select flag is cleared and the next instruction is executed. If AC1 ≠ 0, then the select flag is cleared and AC1 is shifted right until a 1 is shifted out of bit 0. The number of shifts which occurred is added to the contents of AC2. The next memory location is skipped.

JUMP TO LEVEL 0 INTERRUPT, INDIRECT (JINT)

| 15 | | | | 11 | 10 | | | | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | disp | | | |

*Operation:* (STK) ← (PC); (PC) ← ([$120_{16}$ + disp]); INTEN ← 0

*Description:* The contents of the PC are pushed onto the top of the stack. The new contents of the PC are set equal to the contents of the memory location whose address is formed by adding disp to $120_{16}$. The interrupt enable flag is cleared.

### 3.7.6 TRANSFER-OF-CONTROL INSTRUCTIONS

There are two instructions in this group, summarized in table 3–18. The word formats are depicted in figures 3–15 and 3–16.

*Table 3–18. Transfer-of-Control Instructions*

| Instruction | OP2 | Operation | Assembler Format |
|---|---|---|---|
| JUMP THROUGH POINTER | 1010000 | (PC) ← ($100_{16}$ + disp) | JMPP disp |
| JUMP TO SUBROUTINE THROUGH POINTER | 0110 | (STK) ← (PC);<br>(PC) ← (disp + $100_{16}$) | JSRP disp |

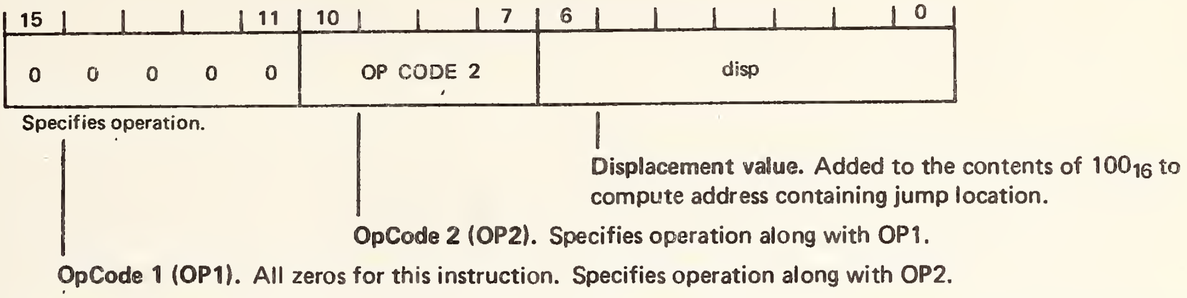

15
11
10
7
6
0
0 0 0 0 0
OP CODE 2
disp
Specifies operation.
Displacement value. Added to the contents of 100$_{16}$ to compute address containing jump location.
OpCode 2 (OP2). Specifies operation along with OP1.
OpCode 1 (OP1). All zeros for this instruction. Specifies operation along with OP2.

*Figure 3–16. Jump to Subroutine Through Pointer Instruction Format*

## JUMP THROUGH POINTER (JMPP)

| 15 | | | | 11 | 10 | | | | | | 4 | 3 | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | disp | | | |

*Operation:* (PC) ← ([100$_{16}$) + disp])

*Description:* The contents of the PC are set equal to the contents of the memory location addressed by the sum of the contents of location 100$_{16}$ and disp.

## JUMP TO SUBROUTINE THROUGH POINTER (JSRP)

| 15 | | | | 11 | 10 | | | 7 | 6 | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | disp | | | | | | |

*Operation:* (STK) ← (PC), (PC) ← ([100$_{16}$) + disp])

*Description:* The program counter is pushed onto the top of the stack. The new contents of the PC are set equal to the contents of the memory location addressed by the sum of the contents of location 100$_{16}$ and disp.

# Chapter 4
# CIRCUIT DESCRIPTIONS

This chapter treats each functional block of the IMP-16C at the circuit level, explaining the circuit operation and design configurations. Functional blocks are considered individually, and, at the end of this chapter, figure 4–6, an overall schematic diagram of the IMP-16C, is presented on three foldout sheets. These should be referred to during the circuit descriptions that follow. Each circuit described is shown in a broken-line enclosure having the name of the circuit.

Figure 4–7 is a functional block diagram of the IMP-16C circuits detailed on the schematic diagram of figure 4–6. Most of the units, the data flow, and the control functions are briefly explained in chapter 2 with reference to figure 2–1. The clock and timing circuits, refresh logic, system initialization, and jump/flag timing and control logic are further details added to figure 4–7 so it corresponds to and gives an exact overview of the actual circuits on the IMP-16C schematic diagram. The sheet number given in each functional block of figure 4–7 refers to the sheet number of figure 4–6 on which the circuit is detailed.

Figure 4–7 is on a foldout sheet located following figure 4–6 so it may be readily referenced from any part of this chapter or as a quick guide to the schematic diagram of figure 4–6.

A parts list and a component layout of the IMP-16C card are presented in table 4–2 and figure 4–8, respectively. (Table 4–2 and figure 4–8 are located on page 4–17/18.)

The descriptions that follow mention one CROM in the text; however, all discussions also apply to the case of two CROMs.

## 4.1 MASTER CLOCK AND 4-PHASE CLOCK GENERATORS *(Sheet 1, figure 4–6)*

The 4-phase clocks required for the CPU devices (one CROM and four RALUs) are generated with a shift register and two MH0026 clock drivers. The master clock signal is generated by a crystal oscillator circuit made from a DM10116 triple line receiver connected as an amplifier and a Schmitt trigger circuit. Two transistors, Q1 and Q2, provide level shifting to convert from ECL levels to TTL compatible logic signals.

The shift register DM74195 generates four clock signals, each of which lasts for two time periods. These signals are then logically gated to yield the odd phases that drive the MH0026 clock driver devices. The MH0026 clock drivers are capable of driving 1000 pf loads with rise and fall times of 20 ns. The typical loading by the CROM and the RALUs is 215 pf (45 pf for each device). The resistors in the output lines of the MOS clock drivers damp out any possible clock overshoots by compensating for the inductance of the clock lines.

The DM74195 shift register outputs are also used to generate some of the other timing signals required in accordance with the timing diagrams shown in figure 4–1. These signals are derived by gating the appropriate shift register outputs with various combinations of the master clock and the shift register input clock.

In figure 4–1, the symbols used to designate clock periods have the following significance: the numbers following the letter C denote the specific time period for which the signal is valid. For example, C23 refers to a clock that is high during T2 and T3. Similarly, CLK81 refers to a signal derived from the logical AND of C81 and CLK. The MOS clocks and phase signals that drive the CPU circuits are shown in figure 4–2.

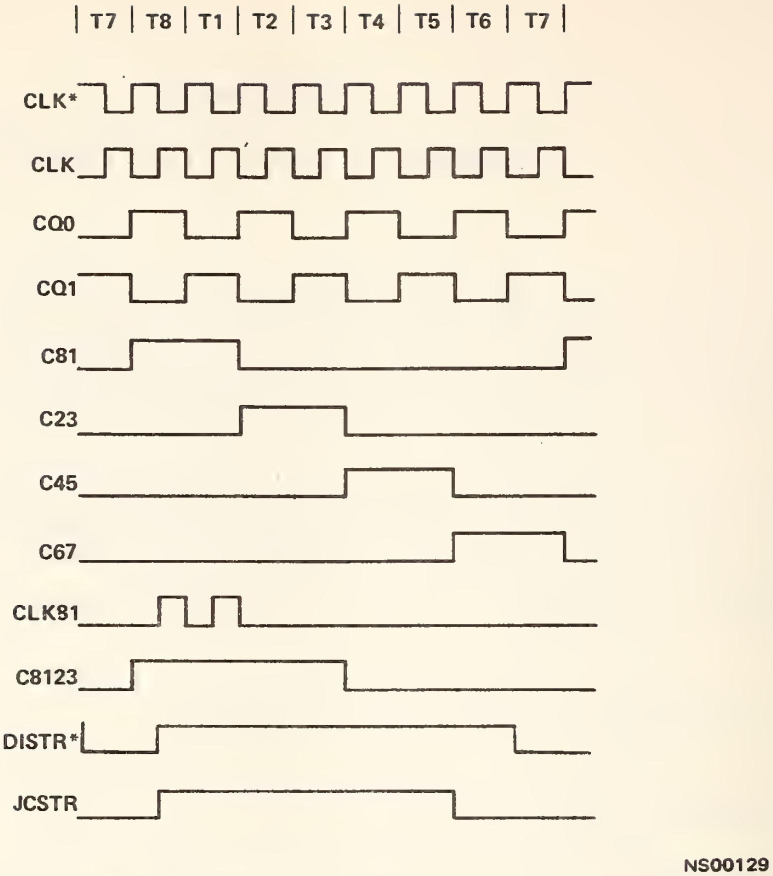


*Figure 4–1. IMP-16C Basic Timing Signals*

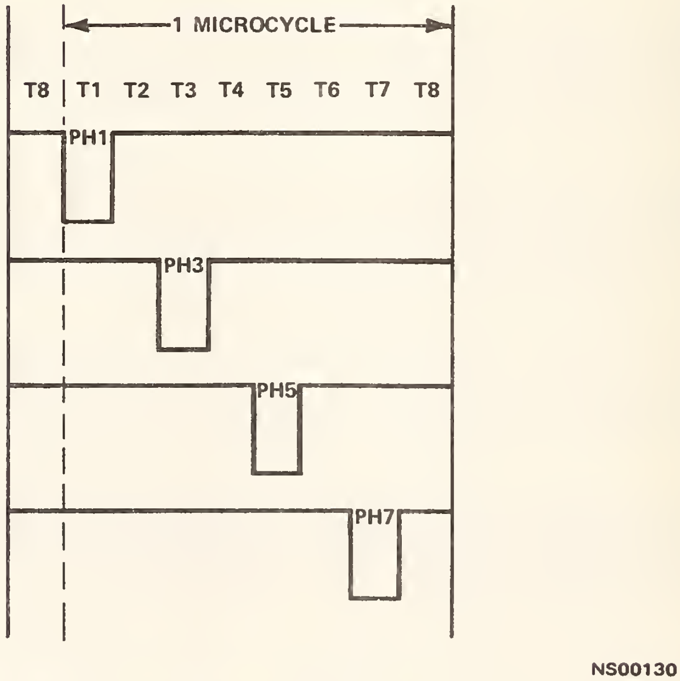


*Figure 4–2. MOS Clocks and Phase Signals*

## 4.2 MOS/LSI CPU LOGIC *(Sheet 3, figure 4–6)*

The CPU consists of one CROM and four RALU circuits driven by the 4-phase clocks. Control between the CROM and the RALUs is effected over the NCB (complemented control bus) lines. The DI (Data In) lines to the CROM serve the purpose of entering the instruction word bits 7 through 15 into the CROM. For sending out a 4-bit address to the Conditional Jump Multiplexer and the Control Flag latches, the lines JFA0 through JFA3 (bidirectional from the CROM) are used. The jump condition signal (NJCOND) enters the CROM at the same pin as bit 7 of the instruction word. The CROM has a flag enable signal[1] (NFLEN) that may be pulsed during T2 to set a particular control flag and/or may be pulsed at T6 to reset the flag (figure 2–4).

The other signals that go to and come from the CPU indicate various status conditions and also perform certain auxiliary operations. The following paragraphs explain these functions.

During an instruction fetch, bits 0 through 7 are loaded into the RALU Memory Data Register (figure 2–2), and bit 7 of the instruction word is extended through bit positions 8 through 15 of the Memory Data Register. In this way, the signed displacement value "disp" discussed in chapter 3 is extended for use in arithmetic operations for forming memory addresses and for immediate instructions. The SININ signal to the RALUs accomplishes the sign extension. For the two low-order RALUs (1 and 2), the SININ pins are permanently connected to a logic 0 (–12V). For the two high-order RALUs (3 and 4), where bits 8 through 15 are located, the SININ pins are connected to the bit 7 output of the Buffered Data Out lines; this bit 7 is used in the two high-order RALUs (3 and 4) to effect the sign extension of "disp" of the instruction word.

The STF signal indicates a "stack-full" condition. When the bottom entry of the stack is filled with nonzero data, the STF line is a "1." The STF lines of all RALUs are tied together and connected to the Conditional Jump Multiplexer to allow testing for the stack-full condition. A similar scheme is used to detect a zero-result condition with the NREQ0 signal. The NREQ0 lines are tied together for all the RALUs; the NREQ0 signal is a "0" if the R-bus is zero as a result of the preceding machine cycle.

During T7 and T8, CSH3 and CSH0 are used to transfer shift data: for a left shift, the most significant bit is shifted out over CSH3, and the least significant bit is shifted in by CSH0; for a right shift, the converse is true.

Each RALU has four status flags, which are interfaced to the A- and R-buses. This provides a convenient means of saving status after an interrupt and for setting the status flags. For all except the most significant RALU (4), the status flags are general purpose and may be used for a variety of functions, depending on the application requirements. For the most significant RALU (4), the status flags have the following functions:

> **LINK Flip-Flop.** When the SEL input to the RALU is "1," the Link Flag (L) is included in shift operations.
>
> **OVERFLOW Flag.** When enabled (under control of the CROM), the Overflow Flag (OV) is set if an arithmetic overflow occurs during an add operation.
>
> **CARRY Flag.** When enabled (under control of the CROM), the Carry Flag (CY) is set to the value of the carry bit out of the most significant ALU bit after an add operation (figure 2–2).
>
> **FLAG Flip-Flop.** This flag is available for general-purpose use.

These status flags may be loaded from the R-bus or stored onto the A-bus under control of the Save/Restore Flag (SVRST) input; this is used by the CROM to implement the PUSHF and PULLF instructions. The output of the general-purpose Flag is available at the Flag output pin; Carry and Overflow Flags are available at CYOV. The Select Flag (SEL) input is used to select the Carry or Overflow for output on CYOV and to determine whether the Link (L) is included in shift operations (discussed in chapter 3). General-purpose flags 0 and 12 are brought out to terminals on the IMP-16C edge connector as signals FLAG0 and FLAG12. The next section describes the flag logic circuits.

[1] The prefix "N" to a signal name denotes logical complementation in the MOS/LSI CROM and RALUs. For signals generated external to these units, an asterisk (*) suffixed to the signal name denotes complementation.

## 4.3 CONTROL FLAGS AND CONDITIONAL JUMP MULTIPLEXER LOGIC *(Sheet 2, figure 4–6)*

External to the CPU portion of the IMP-16C is the logic required to set and reset the control flags and to select one of 16 jump conditions.

The flag addresses sent out by the CROM are latched to keep them stable; this is done with a DM9322 multiplexer connected as a latch by feeding the outputs back to the second set of inputs. (This particular technique has been chosen here because the DM9322 has less propagation delay than conventional DM7475 or DM74175 latches.) The CROM sends out the flag addresses at T1; these are latched in the DM9322 device during the latter half of T1 by the signal CLK81.

The latched addresses are then used to select one of 16 jump conditions in a DM8219 16-to-1 multiplexer. The complemented flag enable signal (NFLEN), which is low at T2 and then again at T6, enables the selection of a flag in one of the two DM9334 8-bit addressable latch devices. The data to the addressable latches comes from the signal C8123, which provides a logic "1" when NFLEN is low at T2 and a logic "0" when NFLEN is low again at T6. This allows setting and resetting of the flags (figure 2–4) under control of NFLEN at T2 and T6, respectively.

The output of the TRI-STATE DM8219 device is tied directly to the jump condition (NJCOND) input of the CROM. This is the line that is tested during conditional jump operations; the testing is done during T2 (figure 2–4). The START and JC12 through JC15 inputs are user-supplied signals and could be asynchronously generated. Thus, in order to ensure that the logic levels for these signals are stable during T2, synchronizing latches are provided.

The various conditions that can be tested are hardwired to the conditional jump multiplexer according to table 3–8 in chapter 3. Four user-assigned jump conditions and six user-assigned control flag lines are brought out to pins on the edge connector.

The more significant 8 of the 16 control flags may be set using the SFLG instruction and cleared or pulsed using the PFLG instruction; the assignments of these flags (F8, INT EN, SEL, and F11 through F15) are listed with their flag codes (FCs) in table 3–13. The SEL control flag affects shift operations as described in 3.6.6, and also selects CY or OV for output on the CYOV line (see table 3–8). The less significant 8 flags are affected by the CROM-resident microprograms. The assignment of these flags is listed in table 4–1.

*Table 4–1. Control Flags Affected by Microprogram*

| Flag Number | Signal Name | Function |
|---|---|---|
| 0 | RDM | Read Memory |
| 1 | WRM | Write Memory |
| 2 | RDP | Read Peripheral |
| 3 | WRP | Write Peripheral |
| 4 | CPINP | Control Panel Input Flag |
| 5 | SVRST | Save/Restore Status Flags |
| 6 | LDAR | Load Address Register |
| 7 | HLT | Set by HALT Instruction |

## 4.4 INPUT MULTIPLEXER, DATA BUFFER, AND ADDRESS LATCHES *(Sheet 3, figure 4–6)*

The 16-bit bidirectional data bus from the RALU devices is used to transfer all information between the CPU and memory and peripheral units. This bus is buffered by passing all output signals through a set of TRI-STATE DM8095 hex-buffer circuits.

Input data destined for the CPU are passed through a set of input multiplexers such that data from a memory or peripheral unit may be switched in. The TRI-STATE DM8123 multiplexers are controlled by RDM, the read memory flag (delayed until T7 because data may be accepted into the CPU only at T7, as shown in figure 2–4).

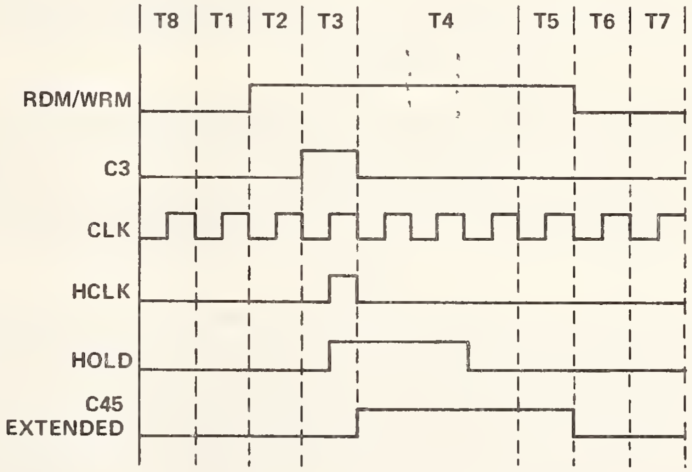
During T7, the data lines are used for input tó the RALU from system memory or peripheral devices. The data receivers are "zeros catching," so the data lines must not be allowed to go negative during T7 unless the data input is to be a zero. Because of this zeros catching feature, the strobe signal for input data (DISTR*) is generated such that it occurs during the latter half of T7; this ensures that the data bits will be strobed in only when it is assured that they are stable.

When the data bus is sending out an address during read and write operations, the address is stored in a register consisting of four DM8551 quad-D flip-flops. These address lines are brought out to terminals on the edge connector to be used for addressing peripheral devices and add-on memory. Data from the RALU are sent out during part of T3 and all of T4 (figure 2–4) and are clocked into the latch if the RDM and the LDAR flag is set. The RDM and LDAR flags are pulsed (set at T2 and reset at T6) during memory read and write operations. The outputs of the DM8551 devices may be disabled by logically controlling the ODIS line. If ODIS is taken to a logic "1," the bus is disabled.

## 4.5 READ/WRITE AND READ-ONLY MEMORIES *(Sheet 4, figure 4–6)*

When executing a memory read operation, the processor sends out an address on the RALU data input/output bus; this address starts coming up during T3, and it is assured of being valid during T4. In the IMP-16C, the address is strobed into a latch at T4. If the processor is used with slow memories whose access times are longer than the interval between T4 and T7, it is necessary to stretch a clock period to allow for the slow access. For this purpose, the circuit clock phase-4 stretcher (sheet 1, figure 4–6) is used to extend T4 for an additional two periods.

During read and write operations, a clock hold signal (HCLK) is developed during C3. The timing relations are given in figure 4–3. This signal sets a flip-flop output (HOLD) such that a count-by-four circuit is enabled. After four counts of the master clock, the HOLD flip-flop is shut off. The delay provided by the counter circuit is used to inhibit phase 4 of the clock generator circuit.



NS00131

*Figure 4–3. Timing Relations for Clock Hold Function*

On-board address decoding for the memory is arranged such that address bit 15 controls the selection of the read-write and read-only memories. Bits 9 through 14 are ignored. Thus, if bit 15 is a logic "1" the ROMs are selected, and if bit 15 is a logic "0" the RWM is selected. If it is desired to change this arrangement or to use add-on memory, external address decoding must be done and appropriate signals supplied to the CS0, CS1, and CS2 pins of the IMP-16C. This option is explained in chapter 9.

## 4.6 INTERRUPT HANDLER *(Sheet 1, figure 4–6)*

The IMP-16C interrupt facility is handled through the conditional jump multiplexer inputs. Two interrupt inputs are provided. One is directly wired to the Conditional Jump Multiplexer and responds to a specific interrupt by jumping to a microprogram subroutine designed for control panel interrupts, as described in chapter 6. The other interrupt is a general interrupt input (INTRA), which can be wired to the user's interrupting device. The processing of interrupts is described in chapter 6.

The flip-flop output (INT-Q1) is set high whenever an external interrupt (INTRA) or a stack-full signal (STFL) is true simultaneously with the interrupt enable (INTEN) flag. INT-Q1 is wired to the interrupt input of the conditional jump multiplexer. The interrupt processing microprogram resets the interrupt enable (INTEN) flag to zero to disable any further interrupts, and control is transferred to the instruction stored in location 1 of main memory. The stack-full line is also wired to the jump condition multiplexer to permit testing for stack-full interrupts. If STFL causes such an interrupt, the bottom entry of the hardware stack is lost. In anticipation of this, the user can put a dummy word in the stack during his program initialization sequences.

See 6.3 for an explanation of the CPINT (Control Panel Interrupt) function.

## 4.7 SYSTEM INITIALIZATION *(Sheet 1, figure 4--6)*

During startup, the IMP-16C is initialized so all the sequential logic is conditioned to known logic states. There are two aspects to initialization: startup of the TTL logic and initialization of the CPU MOS/LSI devices.

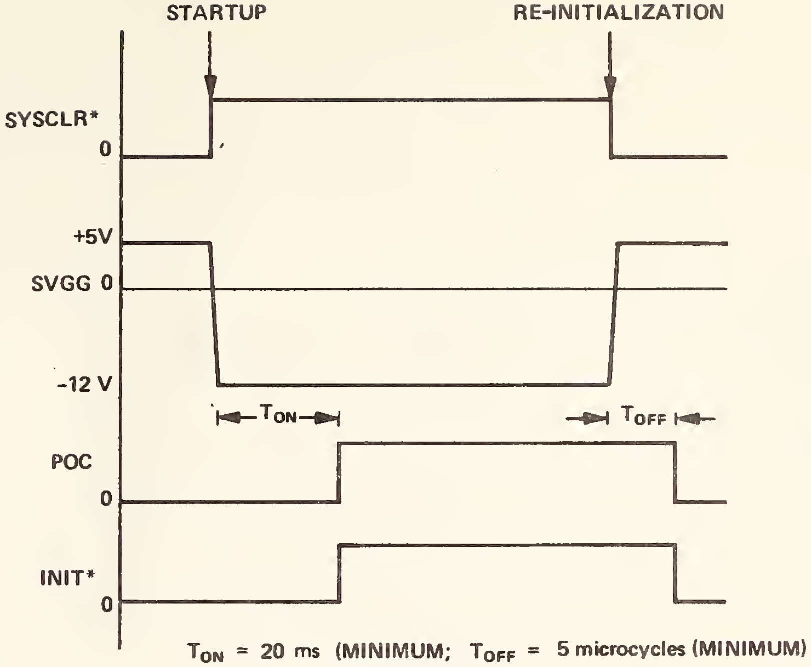
The System Initialization circuit shows the startup logic required for the TTL clocks. When power comes on, the output of flip-flop (INIT*) is forced to a logic "0" (independent of the state of the SYSCLR* input) by the RC timing circuit. Because this signal goes to the Clear inputs of all the other flip-flops in the clock-generator circuit, the system starts up in a cleared condition. The system may also be cleared at any other time by grounding the SYSCLR* input.

When the System Clear signal (SYSCLR*) goes high, INIT* comes up after a delay of a few hundred milliseconds (time constant R10C5). At this time, all the system clocks are enabled. For systems that do not have external initialization, the SYSCLR* input should be left continuously at a logic "1."

Startup for the MOS parts is achieved by controlling the application of the –12-volt supply. The CPU MOS/LSI devices receive –12 volts from the SVGG (switched VGG) line. This voltage must be turned on a few milliseconds before the clocks are started. During turn-off, the clocks are kept on for a few milliseconds after SVGG goes off. The timing relationship between SVGG and the Power-On Condition (POC) signal in the System Initialization circuit is shown in figure 4–4.

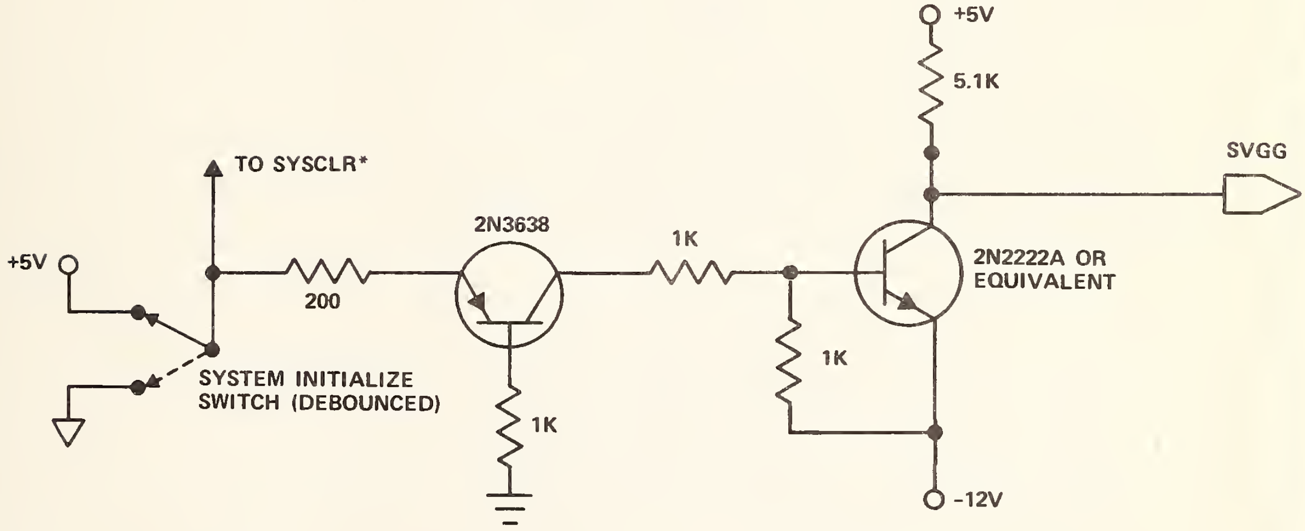
Figure 4–5 is a recommended circuit that may be used to effect system reinitialization to clear the CPU MOS/LSI devices without shutting down other circuits on the IMP-16 card. This circuit is not on earlier versions of the IMP-16C card and would have to be user-supplied and connected to the SVGG terminal pin (sheet 2, figure 4–6). All IMP-16C cards with the part number 5511962 have the initialization circuit on board.

If a special reinitialization circuit, such as discussed above, is not supplied, the SVGG pins on the IMP-16C card-edge connector should be connected to the –12-volt supply. With this latter setup, reinitialization is effected by turning off power to the IMP-16C for at least 5 seconds and then turning it on again.



NS00132

*Figure 4–4. System Startup Timing*



NOTE: BY TYING THE SYSCLR* PIN ON THE IMP-16C CARD TO A "SYSTEM INITIALIZE" BUTTON, THE SAME SIGNAL CAN BE USED TO CONTROL THE SWITCHED -12V (SVGG) FOR THE MOS/LSI DEVICES ON THE IMP-16C CARD. IN THIS CASE, IT IS REQUIRED THAT SYSCLR* BE LOW WHEN POWER IS APPLIED.

NS00133

*Figure 4–5. Circuit for Powering Up CPU MOS/LSI Devices*

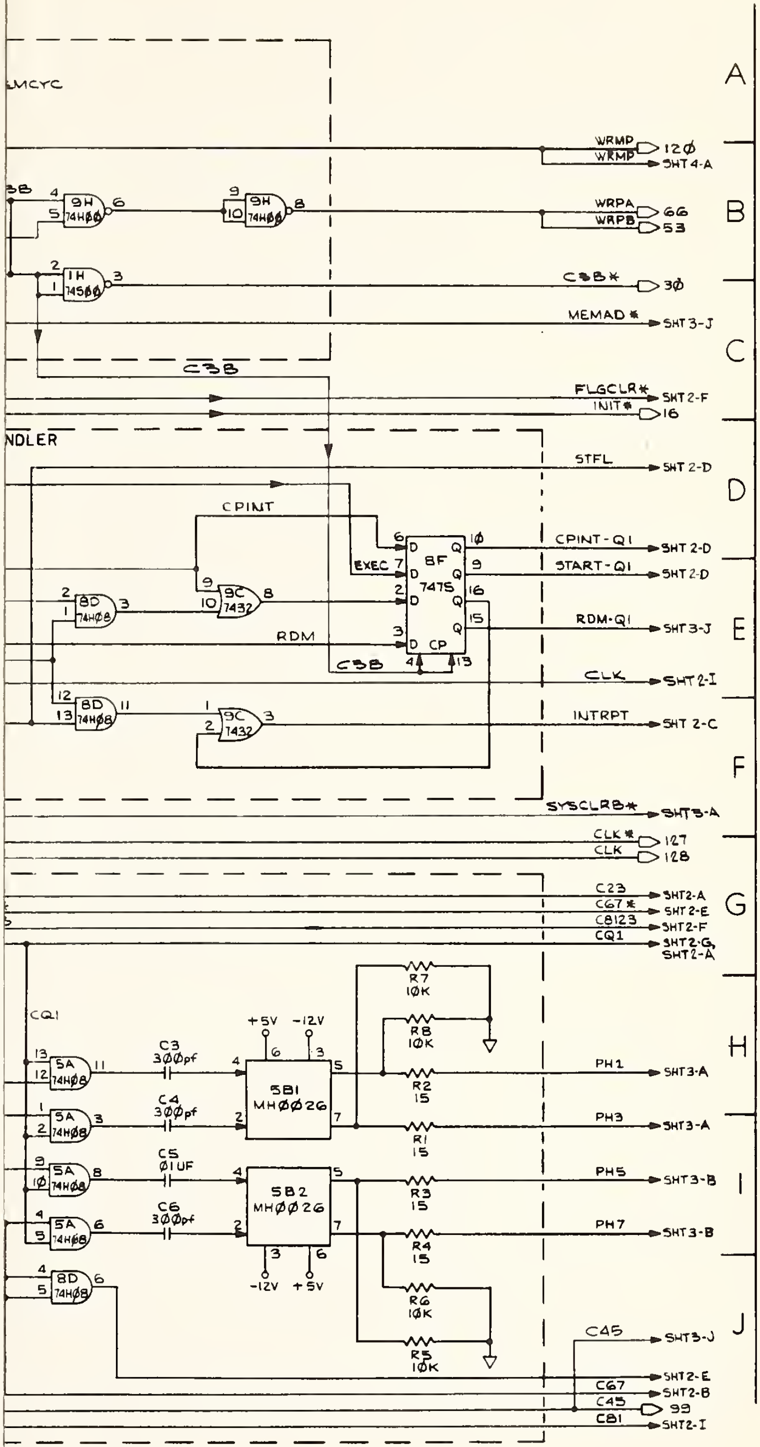
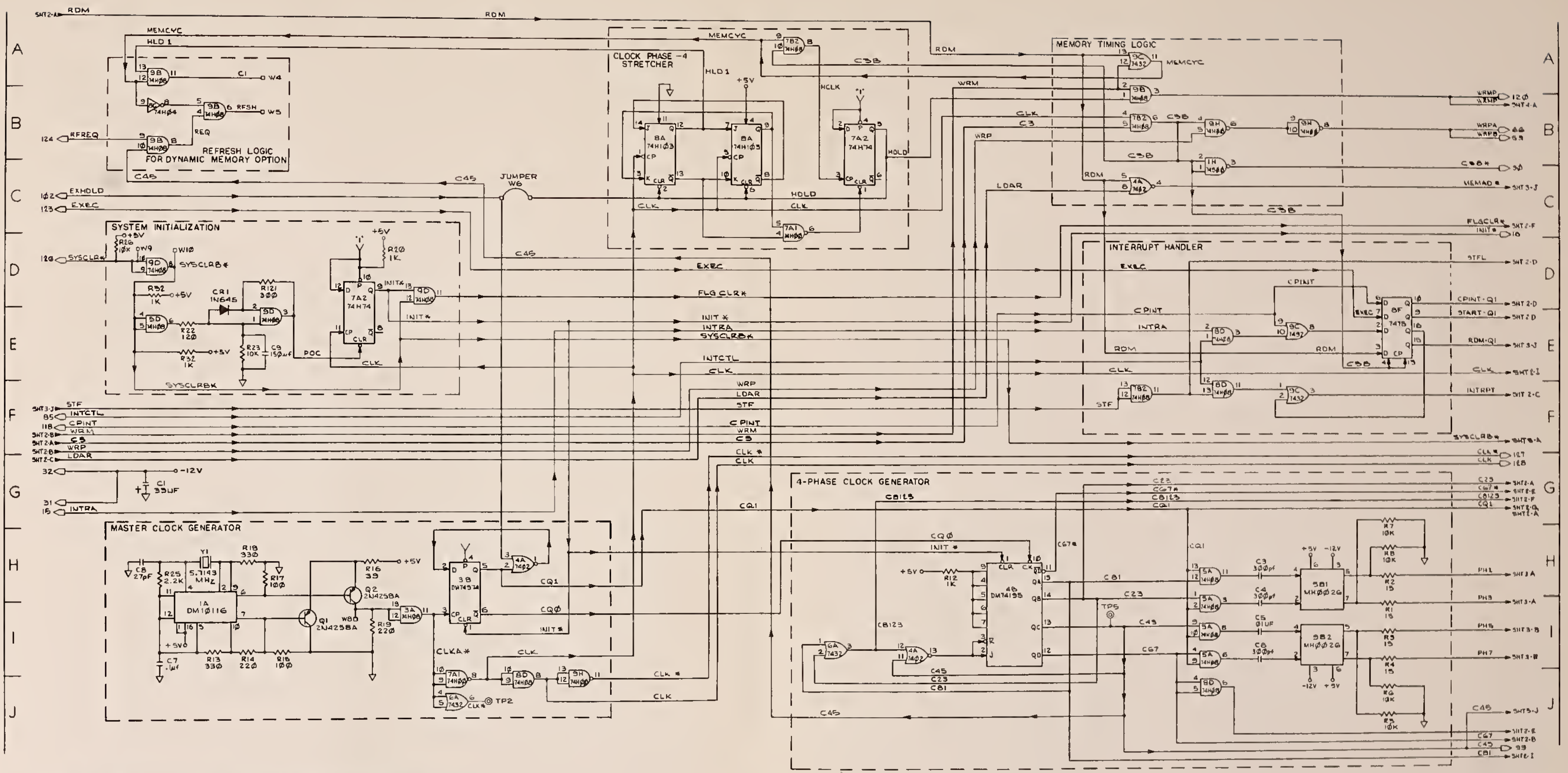


*Figure 4–6. IMP-16C Schematic Diagram*
*(sheet 1 of 4)*



NOTE: INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE. ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

*Figure 4-6. IMP-16C Schematic Diagram (sheet 1 of 4)*

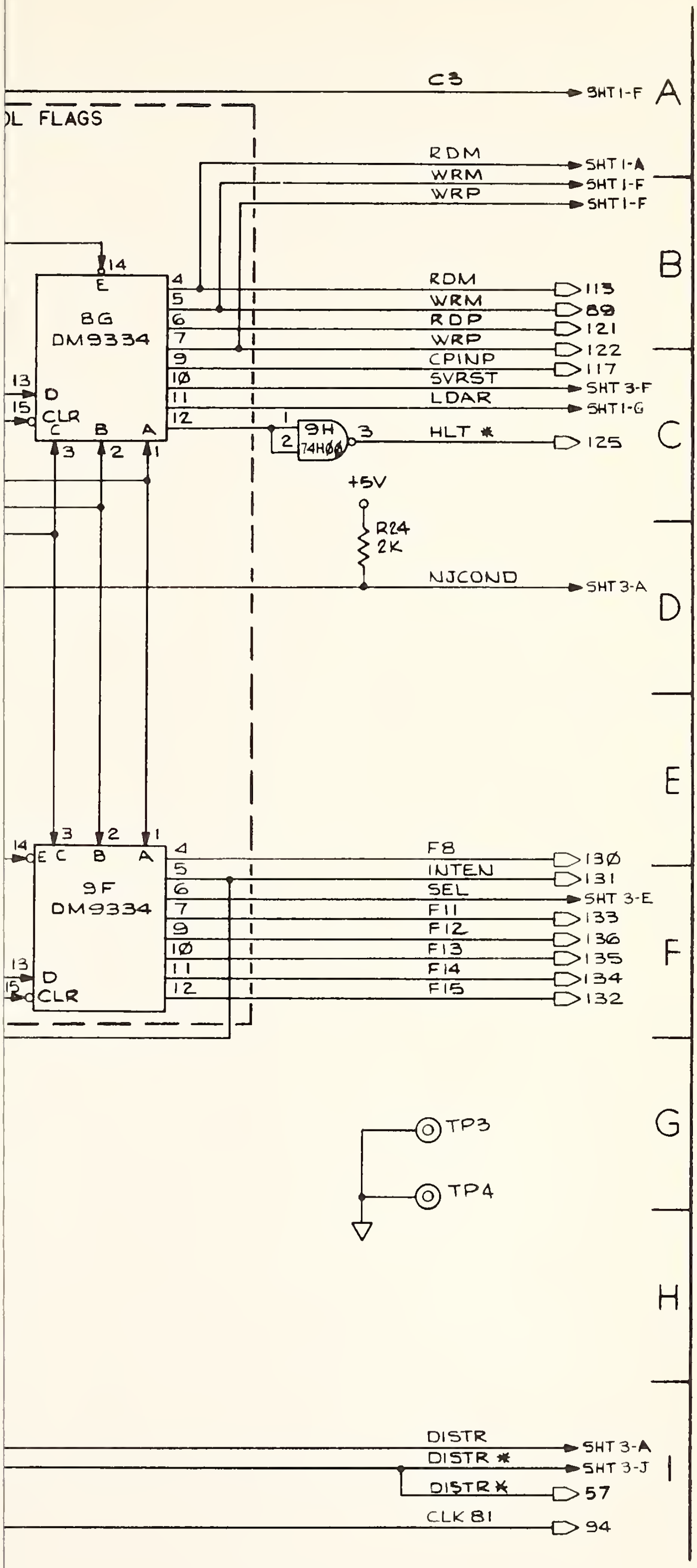
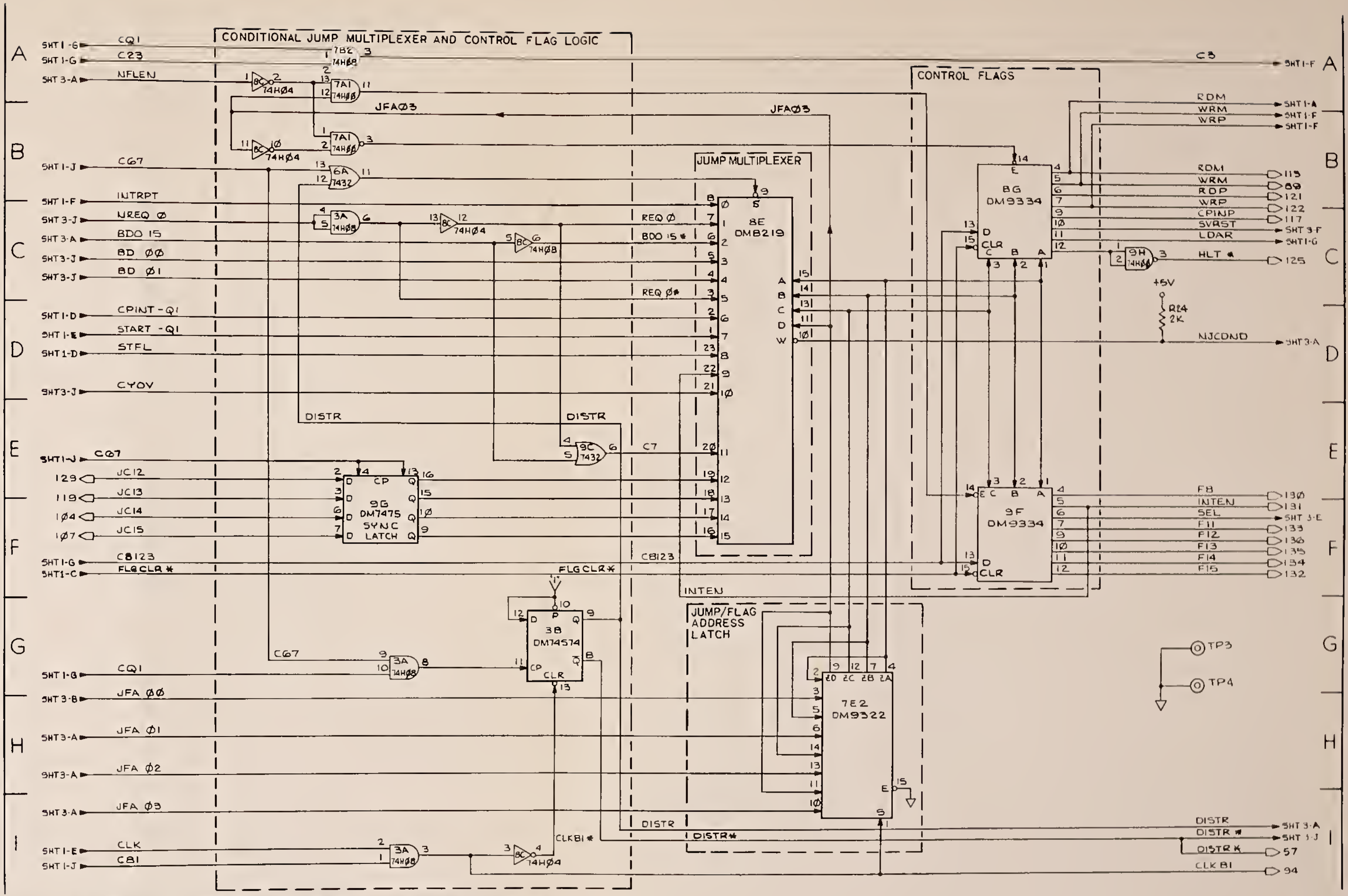


*Figure 4–6. IMP-16C Schematic Diagram (sheet 2 of 4)*



NOTE: INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE. ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

*Figure 4-6. IMP-16C Schematic Diagram (sheet 2 of 4)*

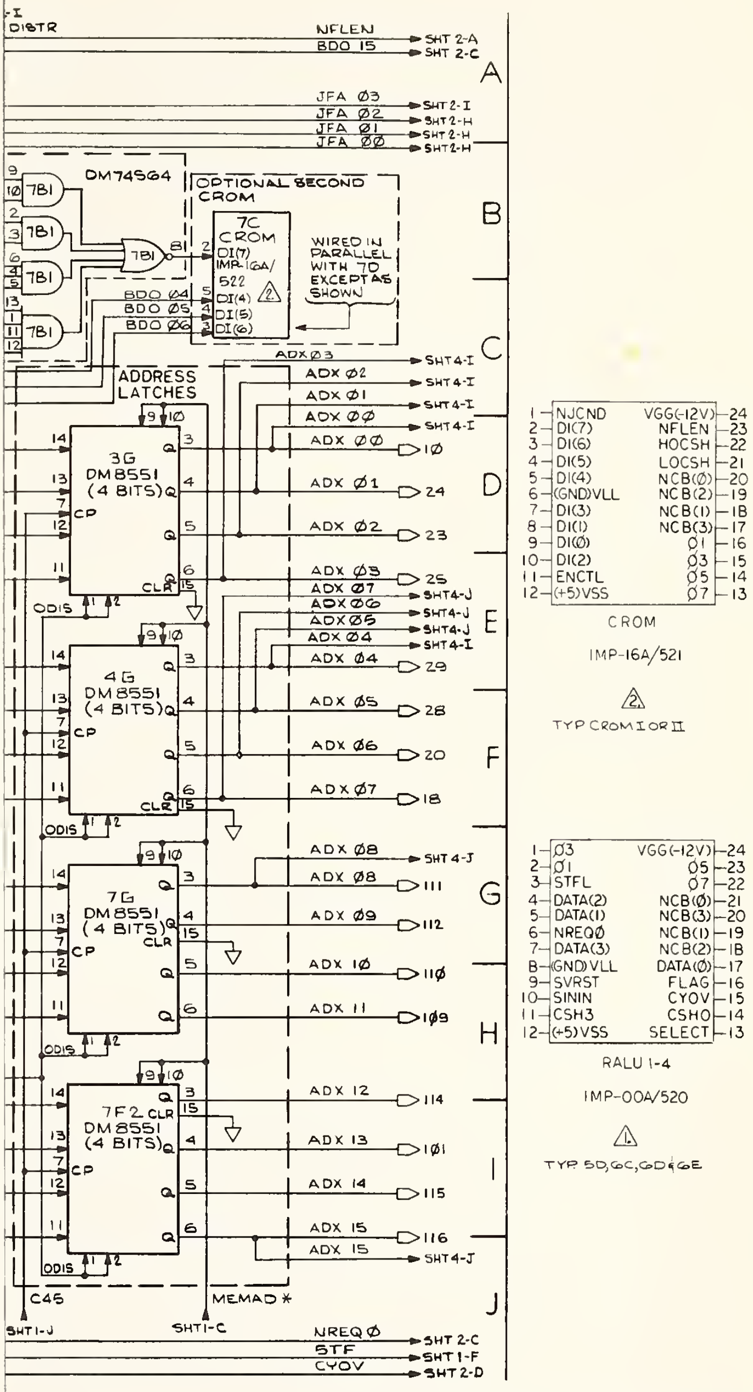
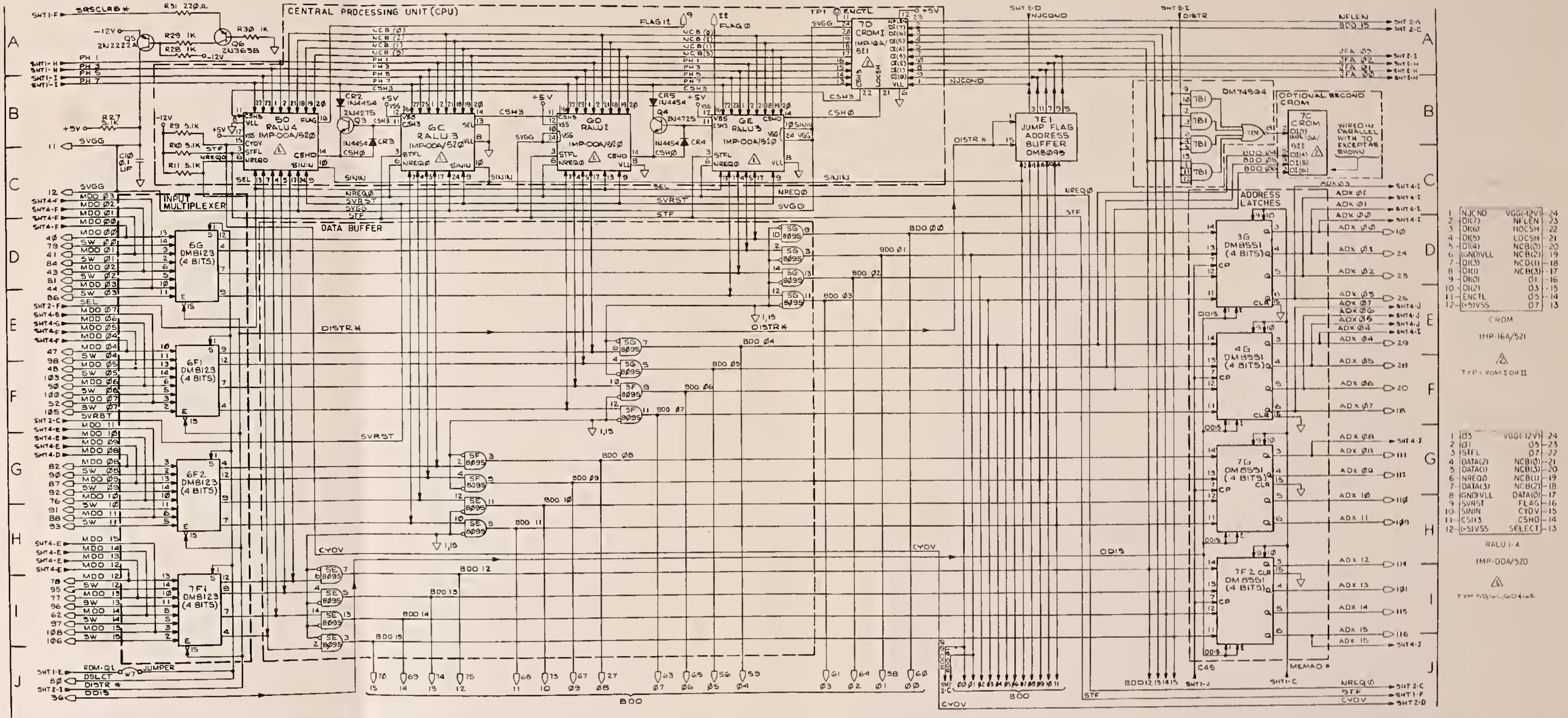
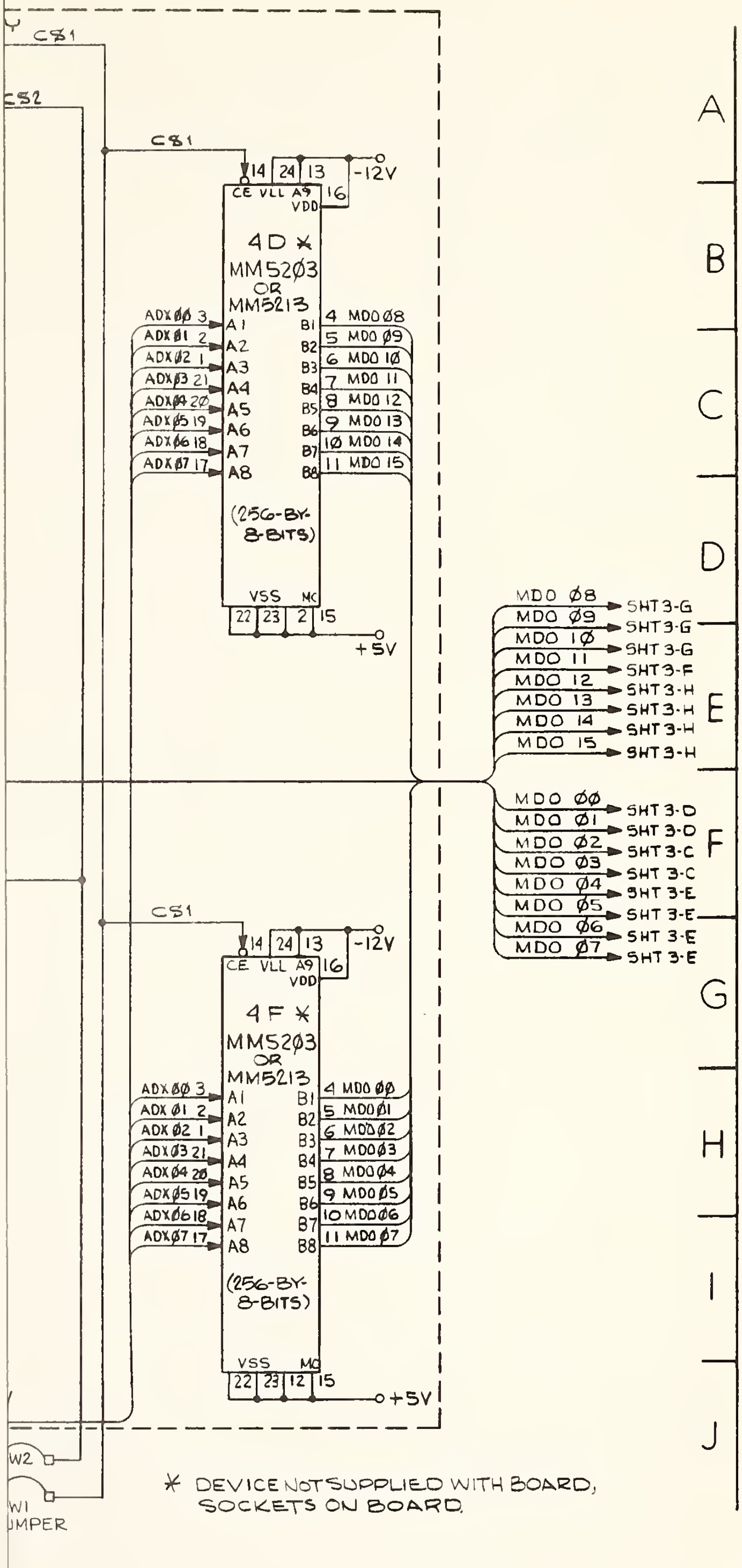


*Figure 4–6. IMP-16C Schematic Diagram (sheet 3 of 4)*



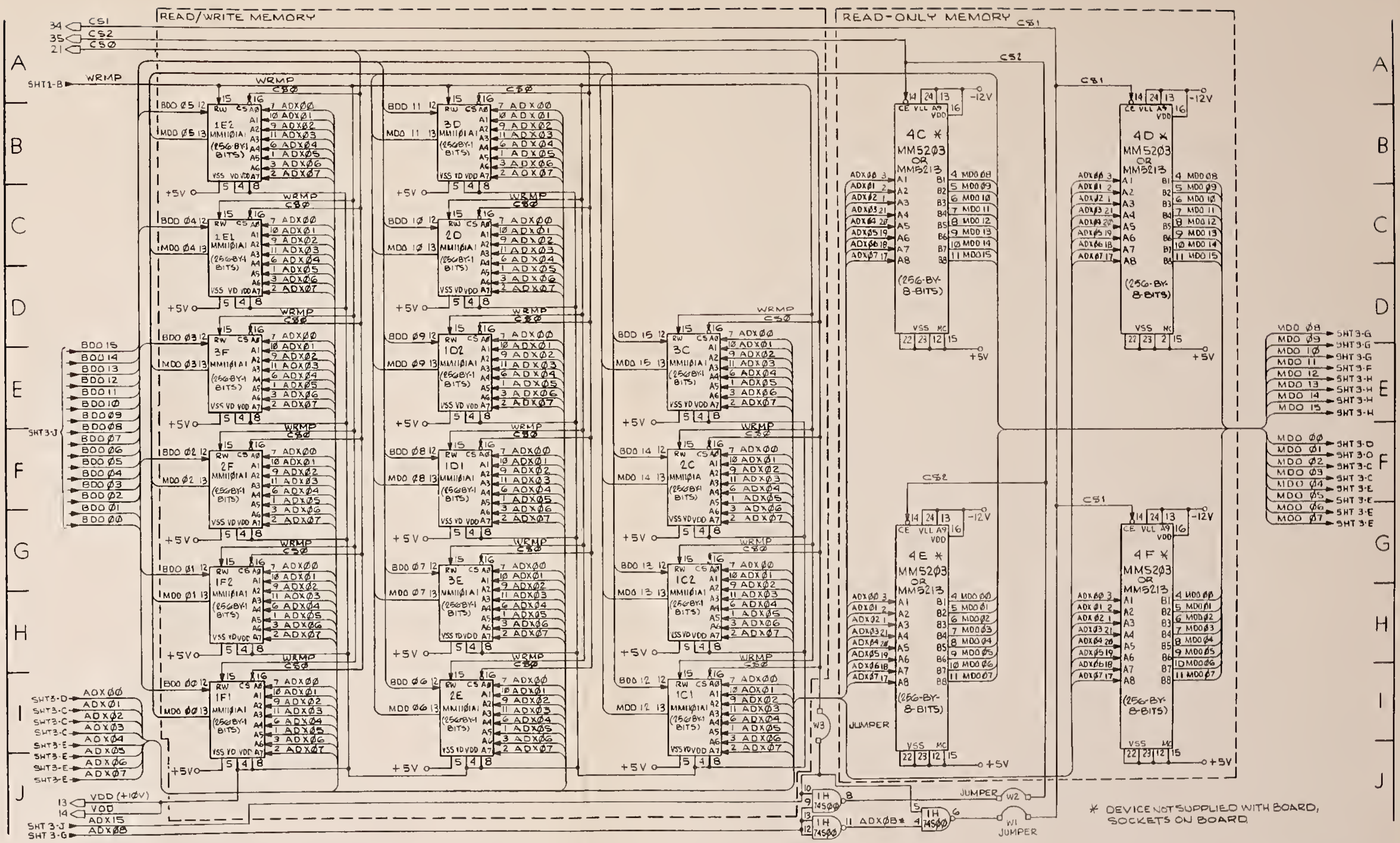
NOTE: INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE. ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

*Figure 4-6. IMP-16C Schematic Diagram (sheet 3 of 4)*



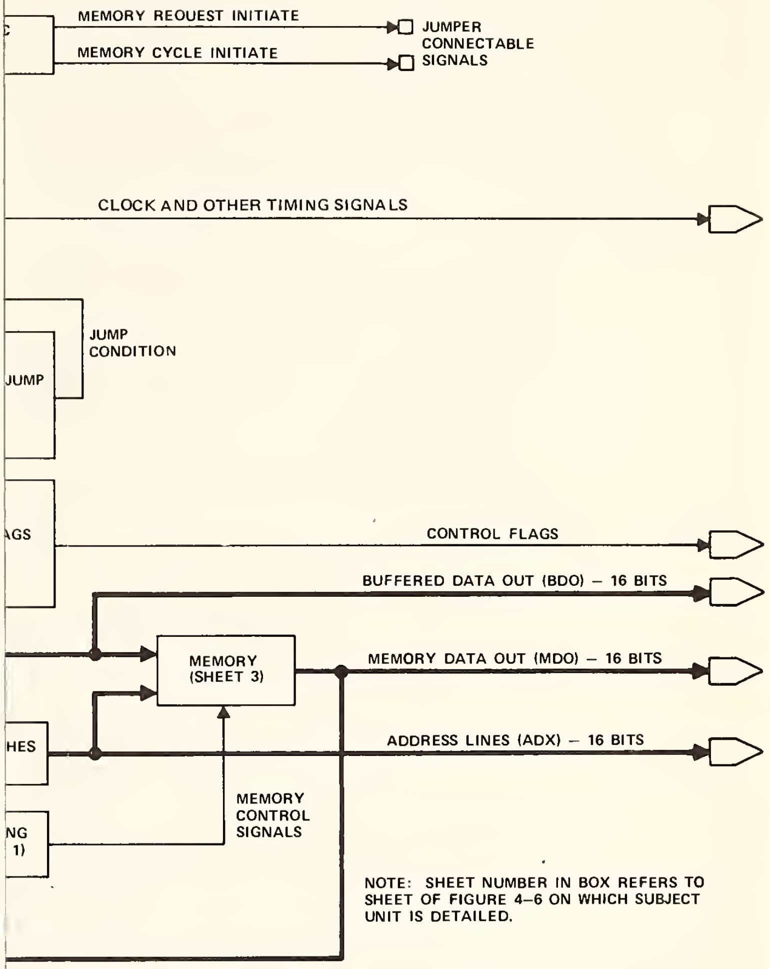
* DEVICE NOT SUPPLIED WITH BOARD, SOCKETS ON BOARD.

*Figure 4–6. IMP-16C Schematic Diagram (sheet 4 of 4)*



NOTE: INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE. ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

*Figure 4-6. IMP-16C Schematic Diagram (sheet 4 of 4)*



C
MEMORY REQUEST INITIATE
MEMORY CYCLE INITIATE
JUMPER CONNECTABLE SIGNALS
CLOCK AND OTHER TIMING SIGNALS
JUMP CONDITION
JUMP
AGS
CONTROL FLAGS
BUFFERED DATA OUT (BDO) – 16 BITS
MEMORY (SHEET 3)
MEMORY DATA OUT (MDO) – 16 BITS
HES
ADDRESS LINES (ADX) – 16 BITS
NG 1)
MEMORY CONTROL SIGNALS

NOTE: SHEET NUMBER IN BOX REFERS TO SHEET OF FIGURE 4–6 ON WHICH SUBJECT UNIT IS DETAILED.

NS00137

*Figure 4–7. IMP-16C Functional Block Diagram*

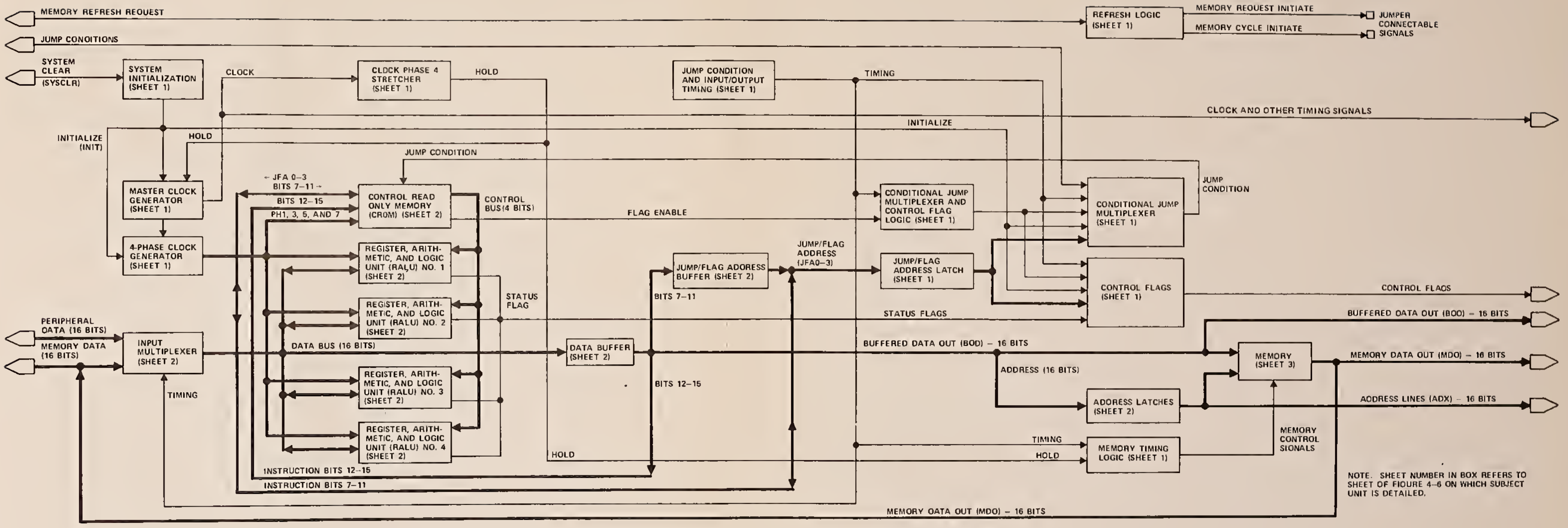
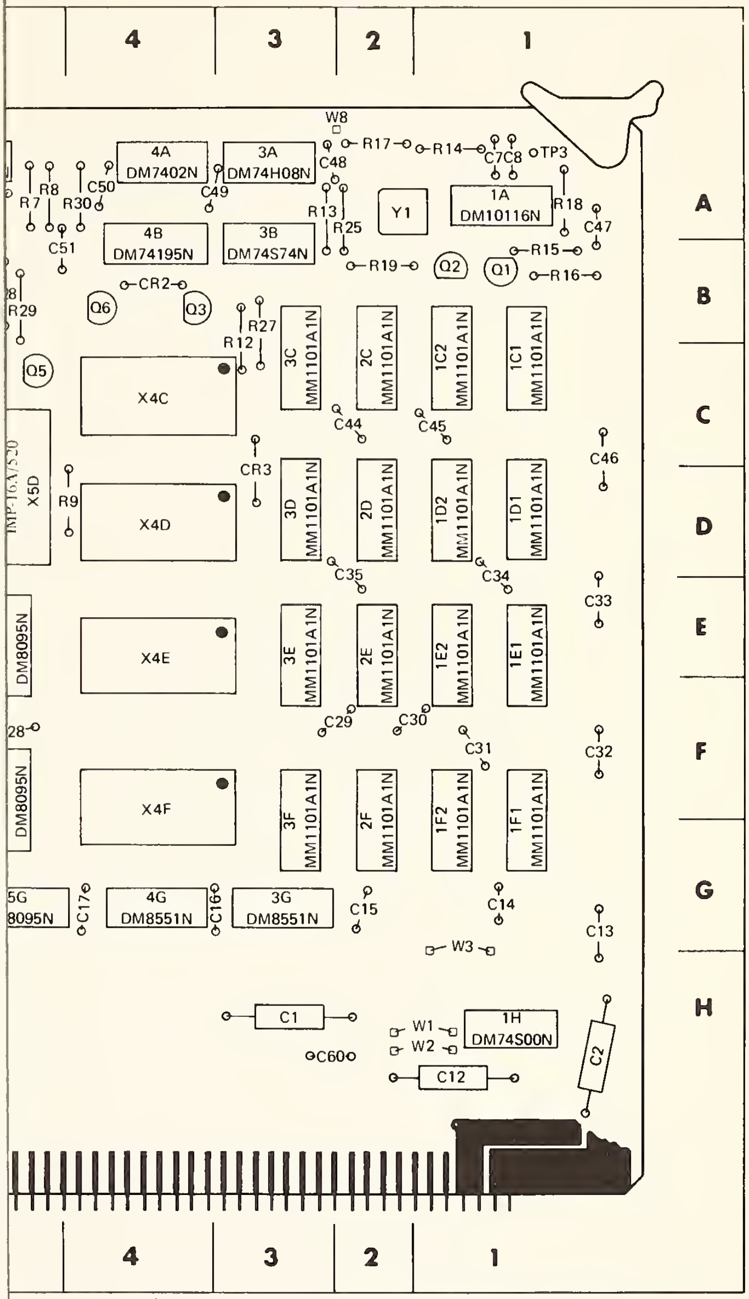

MEMORY REFRESH REQUEST
JUMP CONDITIONS
SYSTEM CLEAR (SYSCLR)
SYSTEM INITIALIZATION (SHEET 1)
CLOCK
CLOCK PHASE 4 STRETCHER (SHEET 1)
HOLD
JUMP CONDITION AND INPUT/OUTPUT TIMING (SHEET 1)
TIMING
REFRESH LOGIC (SHEET 1)
MEMORY REQUEST INITIATE
MEMORY CYCLE INITIATE
JUMPER CONNECTABLE SIGNALS
CLOCK AND OTHER TIMING SIGNALS
INITIALIZE
INITIALIZE (INIT)
HOLD
JUMP CONDITION
← JFA 0–3 BITS 7–11 →
BITS 12–15
PH1, 3, 5, AND 7
MASTER CLOCK GENERATOR (SHEET 1)
4-PHASE CLOCK GENERATOR (SHEET 1)
CONTROL READ ONLY MEMORY (CROM) (SHEET 2)
CONTROL BUS(4 BITS)
FLAG ENABLE
CONDITIONAL JUMP MULTIPLEXER AND CONTROL FLAG LOGIC (SHEET 1)
CONDITIONAL JUMP MULTIPLEXER (SHEET 1)
JUMP CONDITION
REGISTER, ARITHMETIC, AND LOGIC UNIT (RALU) NO. 1 (SHEET 2)
REGISTER, ARITHMETIC, AND LOGIC UNIT (RALU) NO. 2 (SHEET 2)
REGISTER, ARITHMETIC, AND LOGIC UNIT (RALU) NO. 3 (SHEET 2)
REGISTER, ARITHMETIC, AND LOGIC UNIT (RALU) NO. 4 (SHEET 2)
STATUS FLAG
JUMP/FLAG ADDRESS BUFFER (SHEET 2)
JUMP/FLAG ADDRESS (JFA0–3)
JUMP/FLAG ADDRESS LATCH (SHEET 1)
BITS 7–11
CONTROL FLAGS (SHEET 1)
CONTROL FLAGS
STATUS FLAGS
BUFFERED DATA OUT (BDO) – 16 BITS
PERIPHERAL DATA (16 BITS)
MEMORY DATA (16 BITS)
INPUT MULTIPLEXER (SHEET 2)
DATA BUS (16 BITS)
DATA BUFFER (SHEET 2)
BUFFERED DATA OUT (BDO) – 16 BITS
MEMORY (SHEET 3)
MEMORY DATA OUT (MDO) – 16 BITS
ADDRESS (16 BITS)
TIMING
BITS 12–15
ADDRESS LATCHES (SHEET 2)
ADDRESS LINES (ADX) – 16 BITS
MEMORY CONTROL SIGNALS
TIMING
HOLD
HOLD
MEMORY TIMING LOGIC (SHEET 1)
INSTRUCTION BITS 12–15
INSTRUCTION BITS 7–11
MEMORY DATA OUT (MDO) – 16 BITS
NOTE. SHEET NUMBER IN BOX REFERS TO SHEET OF FIGURE 4–6 ON WHICH SUBJECT UNIT IS DETAILED.
NS00137

*Figure 4–7. IMP-16C Functional Block Diagram*

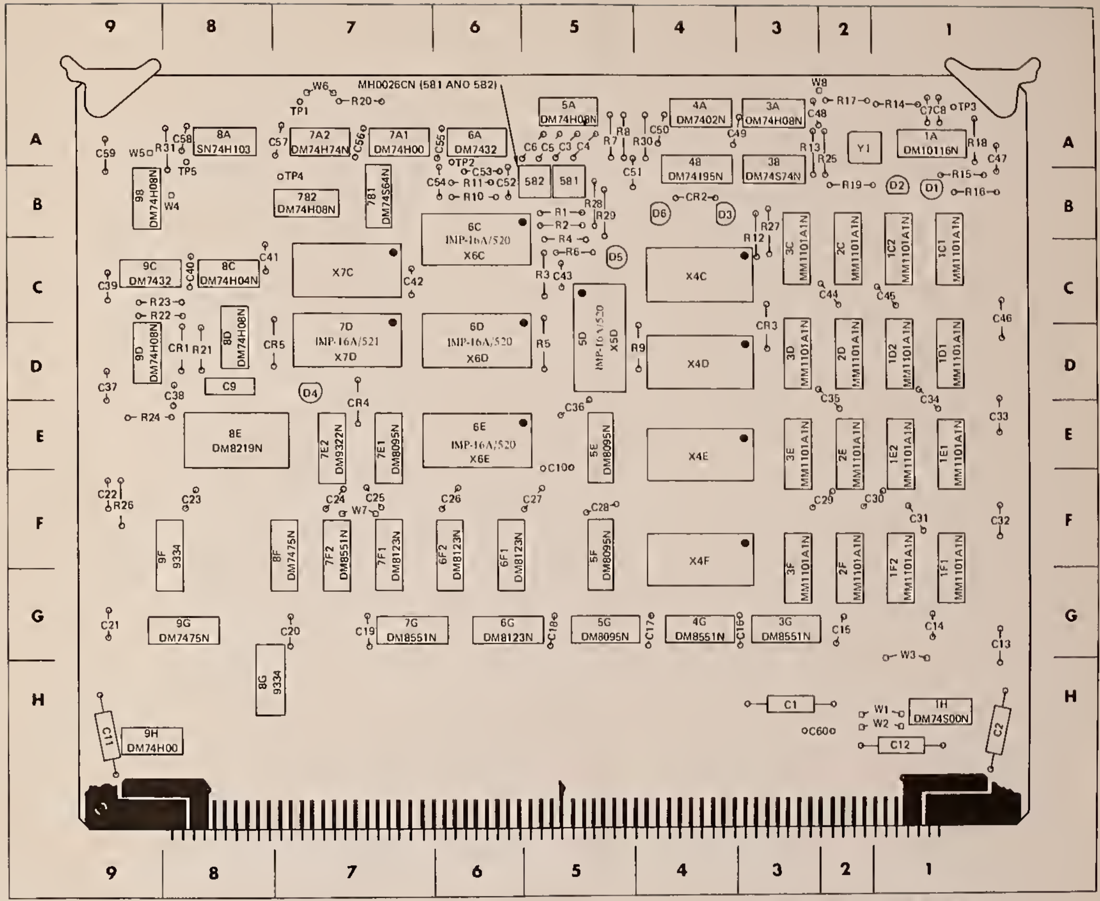


INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE. ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

*Figure 4–8. IMP-16C Card, Component Layout*

*Table 4–2. IMP-16C/200/300 Parts List*

| Item | Description | Reference Designation | Part Number | Quantity |
|---|---|---|---|---|
| | **Capacitors** | | | |
| 1 | Capacitor, 33µf, 20V | C1, C2, C11, C12 | | 4 |
| 2 | Capacitor, 300 pf, 300V | C3, C4, C6 | | 3 |
| 3 | Capacitor, 0.01µf, 50V | C5, C43 | | 2 |
| 4 | Capacitor, 0.1µf, 50V | C7, C10, C13–C42, C44–C60 | | 49 |
| 5 | Capacitor, 27 pf, 500V | C8 | | 1 |
| 6 | Capacitor, 150 µf, 20V | C9 | | 1 |
| | **Crystal** | | | |
| 7 | Crystal, 5.7142 MHz | Y1 | | 1 |
| | **Diodes** | | | |
| 8 | Diode | CR1 | 1N645 | 1 |
| 9 | Diode | CR2–CR5 | 1N4454 | 4 |
| | **Resistors** | | | |
| 10 | Resistor, 15 ohms, ±5%, 1/4W | R1–R4 | | 4 |
| 11 | Resistor, 10K, ±5%, 1/4W | R5–R8, R23 | | 5 |
| 12 | Resistor, 5.1K, ±5%, 1/4W | R9–R11, R27 | | 4 |
| 13 | Resistor, 1K, ±5%, 1/4W | R12, R20, R28–R30, R26 | | 6 |
| 14 | Resistor, 330 ohms, ±5%, 1/4W | R13, R18 | | 2 |
| 15 | Resistor, 220 ohms, ±5%, 1/4W | R14, R19, R31 | | 3 |
| 16 | Resistor, 100 ohms, ±5%, 1/4W | R15, R17 | | 2 |
| 17 | Resistor, 39 ohms, ±5%, 1/4W | R16 | | 1 |
| 18 | Resistor, 300 ohms, ±5%, 1/4W | R21 | | 1 |
| 19 | Resistor, 120 ohms, ±5%, 1/4W | R22 | | 1 |
| 20 | Resistor, 2K, ±5%, 1/4W | R24 | | 1 |
| 21 | Resistor, 2.2K, ±5%, 1/4W | R25 | | 1 |
| | **Transistors** | | | |
| 22 | Transistor | Q1, Q2 | 2N4258A | 2 |
| 23 | Transistor | Q3, Q4 | 2N4275 | 2 |
| 24 | Transistor | Q5 | 2N2222A | 1 |
| 25 | Transistor | Q6 | 2N3638 | 1 |
| | **Integrated Circuits** | | | |
| 26 | MOS/LSI Register, Arithmetic, and Logic Unit (RALU) | 5D, 6C, 6D, 6E | IMP-16A/520 | 4 |
| 27 | MOS/LSI Control Read-Only Memory (CROM) | 7D | IMP-16A/521 | 1 |
| 28 | MOS/LSI Control Read-Only Memory (CROM) – Optional | 7C | IMP-16A/522 | 1 |
| 29 | 256-Bit Static Random Access Memory | 1C1, 1C2, 2C, 3C 1D1, 1D2, 2D, 3D, 2E, 3E, 1E1, 1E2, 2F, 3F, 1F1, 1F2 | MM1101A1 | 16 |
| 30 | Electrically Programmable 2048-Bit Read-Only Memory (PROM) — Optional | 4C, 4D, 4E, 4F | MM5203 | 4 |
| 31 | TRI-STATE®16-Line to 1-Line Multiplexer | 8E | DM8219 | 1 |
| 32 | Schottky – Clamped Transistor-Transistor Logic AND-OR-INVERT | 7B1 | DM74S64 | 1 |
| 33 | Schottky – Clamped Transistor-Transistor Logic Dual D Flip-Flop | 3B | DM74S74 | 1 |
| 34 | Schottky – Clamped Transistor-Transistor Logic Quad NAND Gate | 1H | DM74S00 | 1 |
| 35 | TRI-STATE® Quad D Flip-Flop | 3G, 4G, 7G, 7F2 | DM8551 | 4 |
| 36 | Quad Latch | 8F, 9G | DM7475 | 1 |
| 37 | Quad 2-Input Multiplexer | 7E2 | DM9322 | 1 |
| 38 | Dual D Edge-Triggered Flip-Flop | 7A2 | DM74H74 | 1 |
| 39 | 4-Bit Parallel-Access Shift Register | 4B | DM74195 | 1 |
| 40 | 8-Bit Addressable Latch | 8G, 9F | DM9334 | 2 |
| 41 | Dual JK Edge-Triggered Flip-Flop | 8A | SNH103 | 1 |
| 42 | Quad 2-Input NAND Gate | 7A1, 9H | DM74H00 | 2 |
| 43 | Quad 2-Input OR Gate | 6A, 9C | DM7432 | 2 |
| 44 | Quad 2-Input NOR Gate | 4A | DM7402 | 1 |
| 45 | Quad 2-Input AND Gate | 3A, 5A, 7B2, 8D, 9B, 9D | DM74H08 | 6 |
| 46 | Hex Inverter | 8C | DM74H04 | 1 |
| 47 | Hex Buffer | 5E, 5F, 5G, 7E1 | DM8095 | 4 |
| 48 | TRI-STATE ® Quad 2-Input Multiplexer | 6G, 6F1, 6F2, 7F1 | DM8123 | 4 |
| 49 | Triple Differential Line Receiver | 1A | DM10116 | 1 |
| 50 | 5 MHz 2-Phase MOS Clock Driver | 5B1, 5B2 | MH0026CN | 2 |



NOTES: (1) FOR DEVICES THAT ARE REMOVABLE, A LARGE DOT INDICATES THE LOCATION OF PIN 1
(2) ODD-NUMBERED PINS ARE LOCATED ON COMPONENT SIDE.
(3) EVEN-NUMBERED PINS ARE LOCATED ON SOLDER SIDE.
(4) PREFIX X DENOTES SOCKET.
(5) LOCATIONS X4C, X4D, X4E, X4F AND X7C HAVE SOCKETS ONLY, THESE COMPONENTS TO BE INSERTED BY CUSTOMER.

INFORMATION ON THIS DIAGRAM IS SUBJECT TO CHANGE WITHOUT NOTICE. ENGINEERING DIAGRAMS SUPPLIED WITH IMP-16C SHOULD BE REFERENCED.

*Figure 4–8. IMP-16C Card, Component Layout*

# Chapter 5
# INPUT/OUTPUT OPERATIONS

## 5.1 INPUT/OUTPUT INSTRUCTIONS

Input/output operations are carried out with the RIN (Register In) and ROUT (Register Out) macroinstructions. Functionally, they are similar to the LOAD and STORE instructions in that they address a particular device and initiate data exchanges. The effective address of an input/output device is determined by the sum of the contents of Accumulator 3 (AC3) and the 7-bit control field of the RIN or ROUT instruction. Timing for these instructions is shown in figure 5–2.

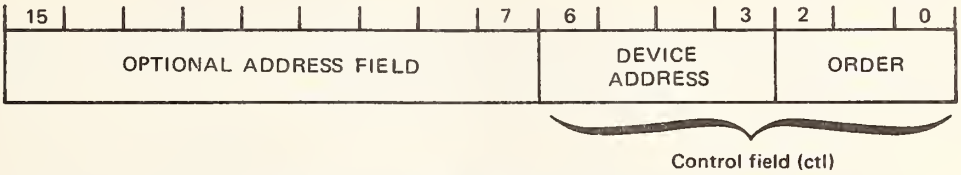


*Figure 5–1. Input/Output Word Format*

Sixteen bits are available for address and command codes. Although a number of schemes are possible, the one described here has proved useful for many applications. The low-order 3 bits may be used to define an input/output "order," and bits 3 to 6 are the device addresses (figure 5–1). Each peripheral device decodes the address field of the input/output instruction command, and if the Read Peripheral or the Write Peripheral flag is active, the device will respond. The 3 "order" bits permit eight possible auxiliary operations for each input/output class; for example, these orders may be read data, read status, reset device, rewind tape, backspace, write data, and so on. The assignment of the various orders is left to the systems programmer.

The 4 bits of the device address field permit direct addressing of 16 devices; however, by loading Accumulator 3 (which is added to bits 0 to 6 of the instruction) with a 16-bit value before executing a RIN or ROUT instruction, up to 65,536 addresses may be specified.

## 5.2 DATA TRANSFER TO PERIPHERAL DEVICES

Peripheral device data may be accessed by the Read Peripheral and Write Peripheral control flags in conjunction with the peripheral device address. Actual transfer of data is effected by the RIN and ROUT instructions during T7 (input) and T4 (output), respectively (figure 2–4). Peripheral device control may be carried out by using the order field of the instruction (figure 5–1) or by dedicating a general-purpose control flag to a peripheral device control function, which can then be controlled by use of the SFLG and PFLG instructions. Similarly, peripheral device status can be sensed by issuing an order to read status over the data bus, or by dedicating one of the general-purpose user jump conditions to this function and using the BOC instruction. (The control flags and jump condition inputs can actually be used to implement a very low-cost serial-data interface without using the data bus at all.)

The functions performed as a result of an input/output command vary. For example, a read peripheral order to a magnetic disc typically initiates a block transfer of information. In contrast, a similar order to a Teletype typically executes the transfer of a single character.

The use of input/output instructions is best illustrated by an example. Consider the case of reading in characters from a serial Teletype unit and transmitting them back immediately (echoing). The following program segment will do this effectively. The first few comment lines define the input/output orders for the Teletype unit. It is assumed that the Teletype is sending data serially over the line corresponding to bit 15.

```
Instruction                         Comment

;                                   DEFINE I/O ORDERS.
;                                   5 = RESET
;                                   2 = READ DATA
;                                   4 = ENABLE TTY PAPER TAPE READER
;                                   3 = WRITE DATA
;                                   TTYAD = TTY ADDRESS IN BIT 3-6
;                                   C1 = 1 (BRANCH IF AC0 = 0)
        START:  LI 3, TTYAD         LOAD AC3 WITH ADDRESS OF TTY.
                ROUT 5;             RESET TELETYPE (ORDER FIELD = 5).
                LI 2, 8             SET COUNT FOR 8 BITS.
        READ:   ROUT 4;             ENABLE TTY READER
                RIN 2;              READ IN TTY DATA
                BOC C2, .+2;        TEST FOR START BIT; C2 REFERS TO AC0 = 0.
                JUMP READ;          READ DATA AGAIN (UNTIL START BIT FOUND).
                JSR DELAY;          START BIT FOUND; DELAY FOR PROPER TIMING.
        INPUT:  ROUT 3;             SEND BIT TO PRINTER.
                JSR DELAY;          DELAY ROUTINE TO TIME OUT 1 BIT
                RIN 2;              READ TTY DATA.
                SHR 0, 1;           SHIFT DATA ONE POSITION.
                AISZ 2, -1;         TEST TO SEE IF DONE
                JMP INPUT;          NOT DONE; READ MORE DATA.
                .                   .
                .                   .
                .                   .
        DELAY:  .;                  START OF DELAY SUBROUTINE
```

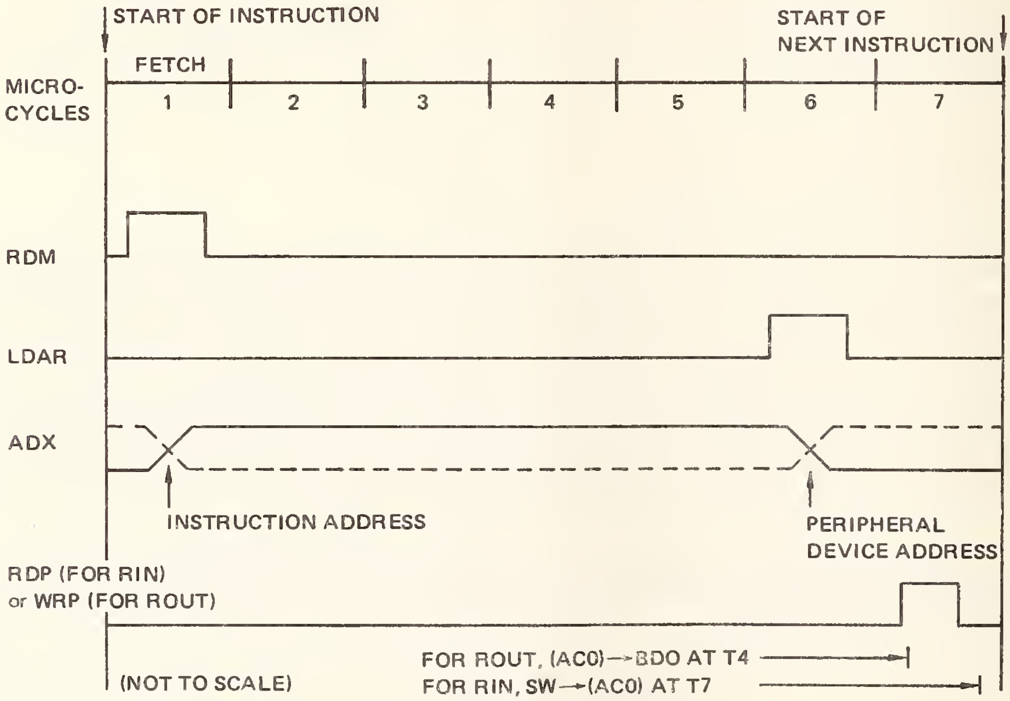


*Figure 5–2. Timing Sequence for RIN and ROUT Instructions*

# Chapter 6
# INTERRUPT SYSTEM

The IMP-16C system recognizes one level of interrupt in its present configuration. A general interrupt request is initiated by the Interrupt Request Signal (INTRA) to the Interrupt Handler (sheet 1, figure 4–6). Also provided is a control panel interrupt input (CPINT). The workings of both types are described below.

## 6.1 GENERAL INTERRUPT

A peripheral device (or any external condition) may send an interrupt request to the IMP-16C over the INTRA line. If the Interrupt Enable Flag is set (that is, no other interrupt currently being serviced), then the interrupt request is latched in a flip-flop and awaits service. During the next instruction fetch cycle, the processor resets the Interrupt Enable Flag (INTEN) and transfers control to location 1 in main memory. At the same time, the PC value is saved on the stack.

The instruction in location 1 of main memory typically would be the start of an interrupt service routine or a jump to a service routine. In the IMP-16C, the stack overflow condition causes an interrupt on the same line. The interrupt service routine can detect this type of interrupt by using a Branch-On Condition (BOC) instruction with cc = 8 (stack full). The interrupt sequence is best illustrated by an example.

## 6.2 EXAMPLE OF INTERRUPT REQUEST AND SERVICE

The case considered here is that of a real-time clock that provides interval timing by sending timed interrupts to the processor. The hardware for this feature would consist of a presettable counter that raises a status signal after it has counted through its sequence. This signal can be used as an interrupt request.

As an example of the use of this timer, consider an application where it is desired to sample a waveform at regular intervals. The real-time clock can be used to generate interrupts at these intervals, and a processing subroutine can read the contents of an analog-to-digital converter driven by the waveform under test. The following program segment shows how this can be done. The clock is assumed to use bit 0 on the data bus to signal its status. This signal is also wired to the general interrupt request line. The interrupt status of the clock is read over bit 0 of the data bus by issuing a READ STATUS order. (Note that other devices could use other bits of the data bus to respond to this order simultaneously. The data bits may then be tested to determine devices that are requesting interrupt service.)

The first few lines of the program are comments that describe the various order codes and addresses assigned to the real-time clock and the A/D converter. The remaining lines of the program segment perform operations according to the requirements above; the comments serve to annotate the program.

```
Label                 Instruction              Comment

;                                              INTERRUPT SEQUENCE FOR REAL-TIME
;                                              CLOCK AND WAVEFORM SAMPLER
;                                              RTC = ADDRESS OF REAL-TIME CLOCK
;                                              ADC = ADDRESS OF A/D CONVERTER
;                                              2 = START CLOCK (I/O ORDER)
;                                              1 = READ DATA ORDER
;                                              0 = READ INTERRUPT STATUS ORDER
                                               C3 = 3; CC FOR BIT 0 OF AC0 = 1
;                                              CLOCK INTERRUPT STATUS HAS BEEN ASSIGNED BIT 0 ON BUS
  LOC1:               JMP     INTR:            THIS INSTRUCTION IS IN LOCATION X'0001
; MAIN PROGRAM FOLLOWS:
                      SFLG    1;               ENABLE INTERRUPT SYSTEM
                              :
                      LI      3, RTC;          LOAD AC3 WITH CLOCK ADDRESS
                      ROUT    2;               START TIMER ON CLOCK
                              :
; SERVICE ROUTINE FOLLOWS:
  INTR:               LI      3, 0;            CLEAR AC3 BEFORE EXECUTING RIN
                      RIN     0;               READ DATA BUS TO CHECK INTERRUPTING DEVICE.
                      BOC     C3, CLKINT;      BRANCH TO CLOCK SERVICE IF BIT 0 = 1.
                              :
; CLOCK SERVICE ROUTINE FOLLOWS:
  CLKINT:             LI      3, RTC;          LOAD AC3 WITH CLOCK ADDRESS
                      ROUT    2;               RESTART TIMER
                      LI      3, ADC;          LOAD ADDRESS OF A/D CONVERTER
                      RIN     1;               READ ADC DATA.
                      JSR     SAMPLE;          GO TO DATA SAMPLE PROCESSING SUBROUTINE.
                      RTI     ;                RETURN FROM INTERRUPT.
                              :
; CONTINUE PROCESSING OF OTHER INTERRUPTS.
                              :
SAMPLE: . . . . . . . . . . . . .;             START OF ROUTINE TO PROCESS DATA FROM A/D CONVERTER.
```

## 6.3 CONTROL PANEL INTERRUPT

The IMP-16C microprogram allows a high-priority interrupt to be indicated on the CPINT line. Since this interrupt is useful for implementing a "program-controlled" control panel, it is called the Control Panel Interrupt; it may, of course, be used for other purposes. If CPINT is active, the processor pulses flag CPINP, reads the data bus at T7 of the same microcycle, and responds as if the data were an instruction. The CPINT input, therefore, can be used to force a jump to some reserved location by "jamming" an instruction on the data lines. This may be used to cause a branch to a control panel service routine or to provide vectored interrupts. By forcing the bus to send in all zeros, the CPINT feature may be used to incorporate a single-step feature, by causing the CPU to halt after the execution of one instruction.

## 6.4 MULTILEVEL INTERRUPTS

It is possible to use the IMP-16C for multilevel interrupt service by use of the RALU general-purpose status flags. Two of these (FLAG0 and FLAG12) are available at the card-edge connectors for this purpose (or other applications where it is desirable to save status). These flags may'be used as interrupt enable flags for two individual levels; they can be modified through use of the PUSHF, PULLF, PULL, and PUSH instructions. A typical arrangement that makes use of the available INTRA input is shown in figure 6–1. (The general-purpose control flags could also be used for this purpose, but their state cannot readily be preserved on the stack as can the status flags.)

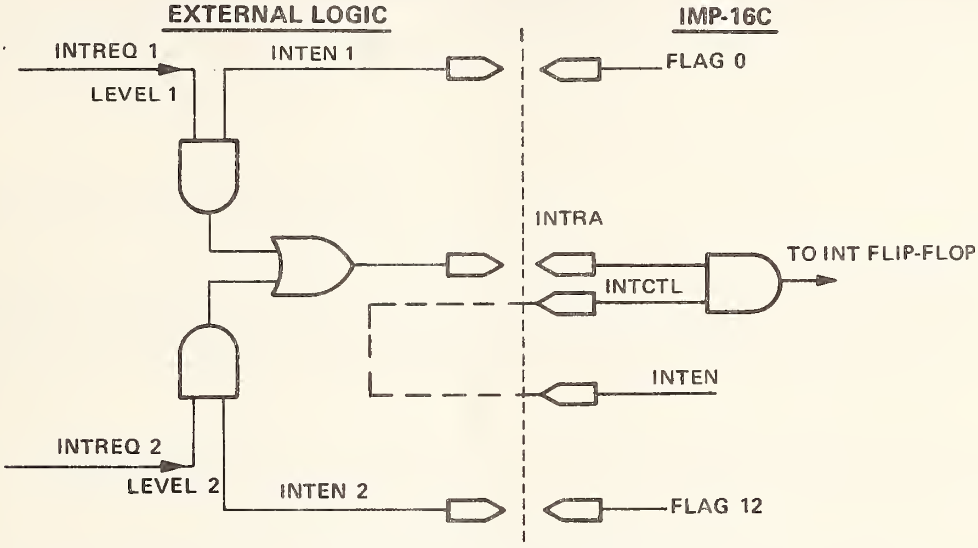


*Figure 6–1. Use of INTRA Input*

The processor responds to inputs on the interrupt request lines that are labeled INTREQ1 and INTREQ2 in figure 6–1. Several devices may be "wire-ORed" to each of these interrupt request lines. If any device generates an interrupt request, the line will go high and interrupt the processor if that level of interrupt is enabled. There is a separate interrupt enable flag for each of the two interrupt levels (labeled INTEN1 and INTEN2) and a master interrupt enable (labeled INTEN) for all levels plus the stack overflow interrupt. The interrupt enable flags for each individual level are part of the CPU status flags, and, as mentioned above, they can be modified by use of the PUSHF, PULLF, PULL, and PUSH instructions. The INTEN flag is one of the IMP-16C control flags. It is modified by use of the Set Flag (SFLG) and the Pulse Flag (PFLG) instructions.

The availability of several different interrupt levels provides a convenient means of controlling interrupts for devices of different priority in a system with a number of peripherals. When only one interrupt request line is used, the interrupt service program must issue an order to each device of lower priority than the one being serviced to reset the lower priority interrupt enable flags on the peripheral controllers. This could be very time consuming in a system with many peripherals. In a system with several interrupt levels, all devices of like priority are tied to the same interrupt request line. For all devices on an individual level that have a common interrupt enable flag at the processor, the interrupt enable flag can disable all devices on that level simultaneously. When an interrupt occurs, lower priority levels are therefore disabled in a minimum amount of time.

The interrupt handling process is summarized by the following. When the processor responds to an interrupt request, it performs the following tasks in order to transfer program control to the interrupt service routine:

- Checks the type of interrupt; if caused by CPINT, jumps to CPINT service routine.
- Transfers the contents of the Program Counter (PC) to the top of the stack.
- Places the address of memory location 1 into the PC.

- Disables (clears) the processor's Interrupt Enable (INTEN) flag to prevent further interrupt.
- Fetches and executes the next instruction from memory location 1, thus initiating the interrupt service routine.

The CPINT service routine does the following:

- Pulses the CPINP flag, thereby indicating that a control panel interrupt is active.
- Reads the data bus and treats the incoming data as an instruction to be executed.

# Chapter 7
# CONTROL PANEL AND 4K-BY-16 MEMORY

The IMP-16C is self-contained for controller applications. However, for some general-purpose applications, a control panel may be needed or convenient to have for debugging, verifying operation, troubleshooting, and other operator-controlled uses. Also, additional memory capacity may be required, and this is available in the National Semiconductor 4K-by-16 random access memory modules.

## 7.1 CONTROL PANEL OPERATION

The simplest control panel for use with the IMP-16C is one that is serviced by software and has a minimum of extra components. Such a panel would consist of a set of 16 data switches, 16 indicator lights, and a few active function switches. These are represented schematically in figure 7–1. The Light-Emitting Diode (LED) devices are driven by a set of DM7475 latches that are strobed by a WRITE PERIPHERAL flag pulse; in this mode, the lights are considered to be an output device.

The spare jump condition multiplexer inputs on the IMP-16C may be tied to active switches on the control panel. These switches may be used for controlling operations such as Load Address, Load Data, Execute, Display Data, and others.

A software service routine that resides in main memory can scan these switches using the Branch-On Condition (BOC) instruction. If any switch is active, an appropriate service action may be initiated. The following program segment is an example of such a routine. It assumes that jump conditions 12, 13, and 7 are wired to switches that cause Load Address, Load Data, and Execute operations to occur. This simple example assumes that there is no peripheral address decoding; all input/output operations are activated by READ PERIPHERAL and WRITE PERIPHERAL control flags as commanded by RIN and ROUT instructions.

```
C12 = 12        ;     LOAD ADDRESS CONDITION.
C13 = 13        ;     LOAD DATA CONDITION.
C7  = 7         ;     EXECUTE
 =X'FFFE        ;     ASSEMBLER DIRECTIVE TO PLACE NEXT INSTRUCTION
                ;       AT LOCATION X'FFFE.
JMP BEGIN
 .
 .
 .
```

*Program continues on page 7–3.*

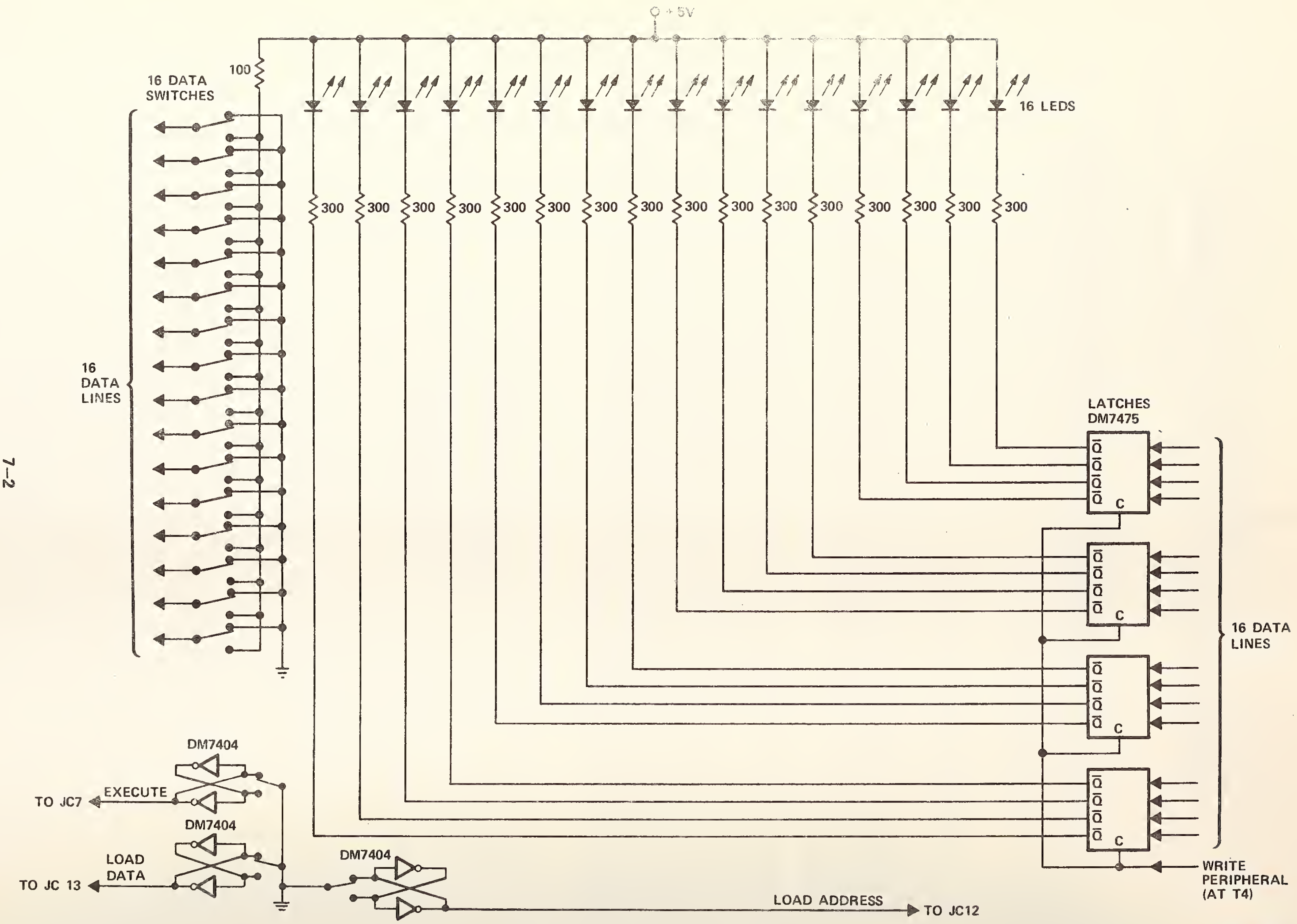


*Figure 7–1. Control Panel Example*

```
BEGIN:    BOC    C12, LA    ;    WAIT LOOP TO
          BOC    C13, LD    ;    SCAN PANEL
          BOC    C7, EX     ;    FUNCTION SWITCHES.
          JMP    BEGIN      ;
LA:       BOC    C12, LA    ;    SWITCH "DEBOUNCER".
          RIN               ;    READ SWITCHES.
          RCPY   0, 2       ;    SAVE ADDRESS IN AC2.
OUT:      ROUT              ;    DISPLAY ADDRESS.
          JMP    BEGIN      ;    RETURN TO WAIT LOOP.
LD:       BOC    C13, LD    ;
          RIN               ;    READ SWITCHES.
          ST     0, (2)     ;    STORE DATA IN ADDRESSED LOCATION.
          ROUT              ;    DISPLAY DATA.
          AISZ   2, 1       ;    INCREMENT ADDRESS.
          JMP    BEGIN      ;    RETURN TO WAIT LOOP.
          HALT              ;    END OF ADDRESS RANGE.
EX:       BOC    C7, EX     ;
          RIN               ;    READ SWITCHES.
          RCPY   0, 2       ;
          JMP    (2)        ;    JUMP TO STARTING ADDRESS.
          .
          .
          .
```

The program above is an example of how the jump condition inputs can be used to scan external conditions and/or requests.

## 7.2 DYNAMIC MEMORY INTERFACE

A circuit has been provided on the IMP-16C card to enable interfacing with the National Semiconductor 4K-by-16 dynamic memory modules. This circuit consists of logic that can receive memory refresh request and send out appropriate refresh orders and read/write cycle initiate signals. The refresh logic appears on the IMP-16C schematic diagram (figure 4–6) but is repeated in simplified form in figure 7–2.

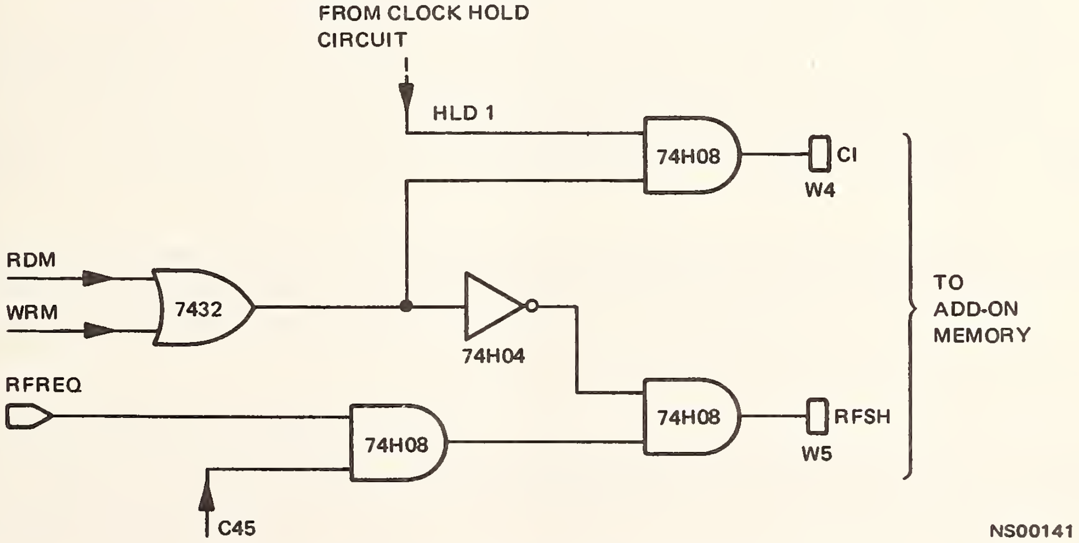
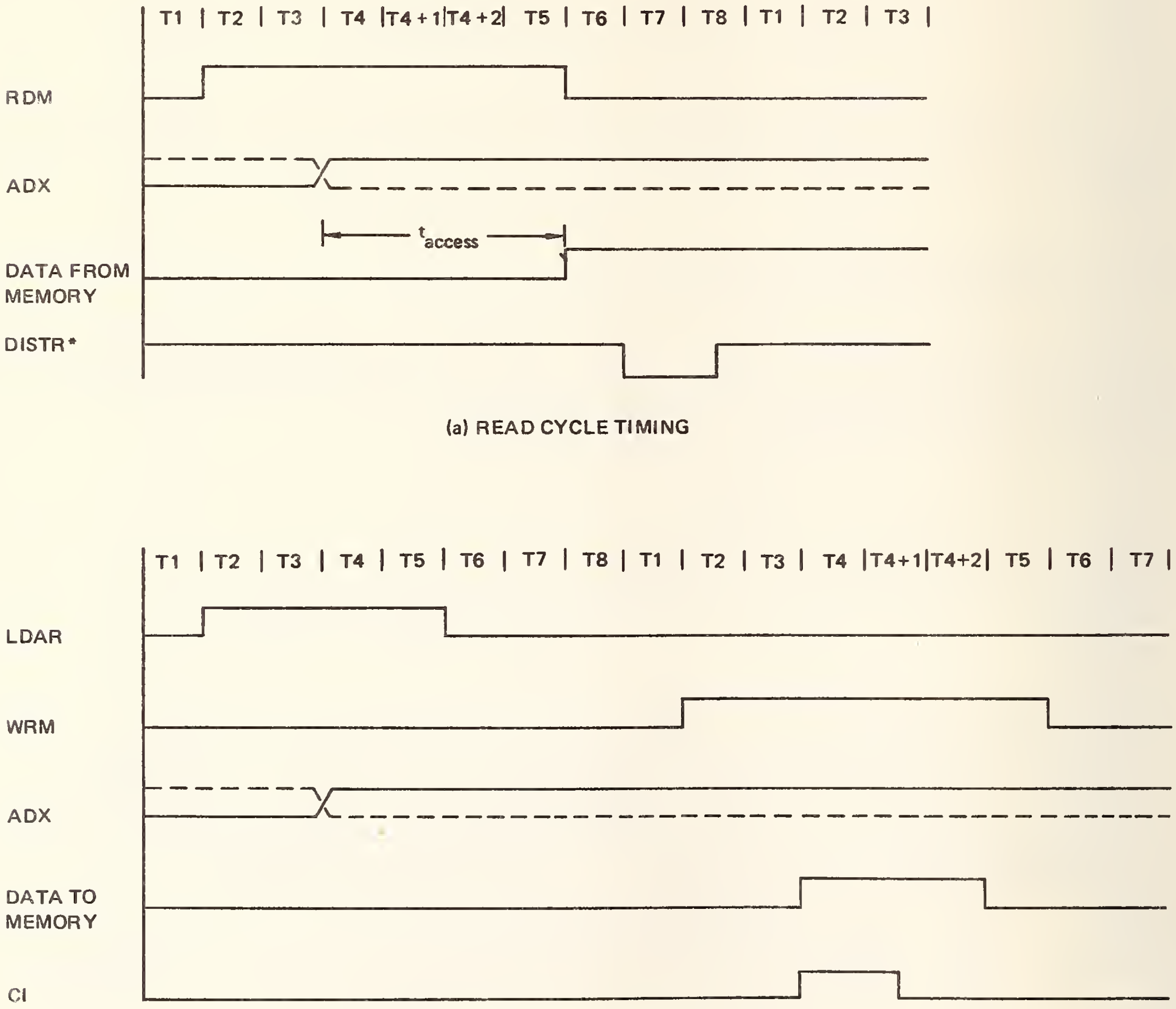


*Figure 7–2. Refresh Logic for Dynamic Memory*

The 4K-by-16 memory requires a refresh cycle of 1 μs duration every 60 μs. The IMP-16C makes a memory reference during every fetch operation and a few other times in-between, depending on the instruction being executed. If the processor is busy during a read or write operation, the memory waits for a refresh cycle until the processor is free. The refresh request signal (RFREQ) is gated into the circuitry at T4. If, at this time, a memory read/write is not requested, an RFSH signal is sent to the memory; otherwise, a Memory Cycle Initiate (CI) signal is sent to memory. Timing for memory read and write operations is given in figure 7–3.

The memory option is provided in the form of a jumper-connectable circuit. If used, jumper pads W4 and W5 may be connected to any two of the unused pins on the edge connector of the IMP-16C.

If add-on memory is attached to the IMP-16C, the user must be aware that the added capacitive loading will slow down the memory access time. Typically, if more than 8K of additional memory is used, the clock periods must be stretched to provide extra time. This may be done using the external hold input which is described in chapter 9.


T1 | T2 | T3 | T4 | T4+1 | T4+2 | T5 | T6 | T7 | T8 | T1 | T2 | T3
RDM
ADX
t_access
DATA FROM MEMORY
DISTR*
T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T1 | T2 | T3 | T4 | T4+1 | T4+2 | T5 | T6 | T7
LDAR
WRM
ADX
DATA TO MEMORY
CI

(a) READ CYCLE TIMING

(b) WRITE CYCLE TIMING

*Figure 7–3. Memory Timing Waveforms*

# Chapter 8
# SYSTEM VERIFICATION

## 8.1 INTRODUCTION

The IMP-16C is a large-scale integrated (LSI) processor consisting of several components from different logic families and types. As such, the proper functioning of the IMP-16C is dependent on the interfacing and timing between the various components. In order to facilitate hardware debugging, a brief system verification procedure is described in this chapter to aid users in the understanding of the system.

For system verification, the IMP-16C may be divided into four distinct partitions:

- Timing and Clocks
- MOS CPU Logic
- Control Flags and Logic
- Data Buses

## 8.2 TIMING AND CLOCKS

When the system is started with the initialization sequence described in chapter 4, the system initialize signal (INIT*) enables all the timing signals. The clocks start with phase 1 and generate all the other auxiliary timing signals. Figure 8–1 shows all the clock signals and their timing relationships. After it has been verified that the MOS clocks are being generated properly, the MOS CPU logic can be checked out.

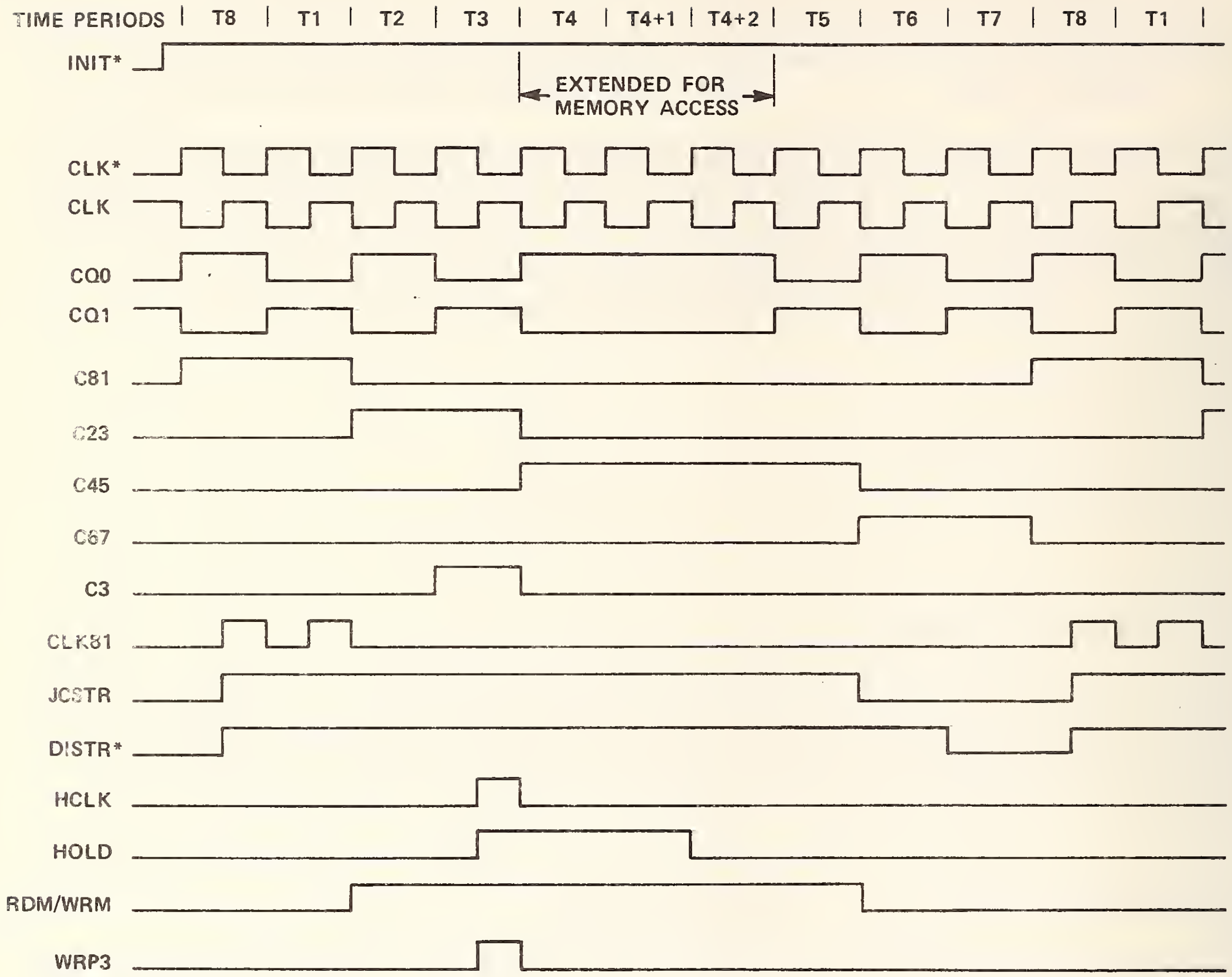
## 8.3 MOS CPU LOGIC

If the system is functioning properly, the CPU executes instructions stored in external memory or executes an instruction(s) from a switch register. In either case, if instructions are being executed, the CROM cycles through repeated FETCH operations. By monitoring the CROM ENCTL line (see sheet 2, figure 4–6), this fact can be verified. The ENCTL line is high between T2 and T7 of every cycle immediately preceding a fetch (except for multiple shift/rotate operations). If the number of microcycles between successive ENCTL pulses is counted for each instruction and checked against table A–1 in appendix A, then it can be assumed that at least most of the CROM circuits are functioning (see figure 8–2).

If the system is being used with some form of control panel (as, for example, the one described in chapter 7), then the RALU can be checked out by outputting register values to a set of display lights.

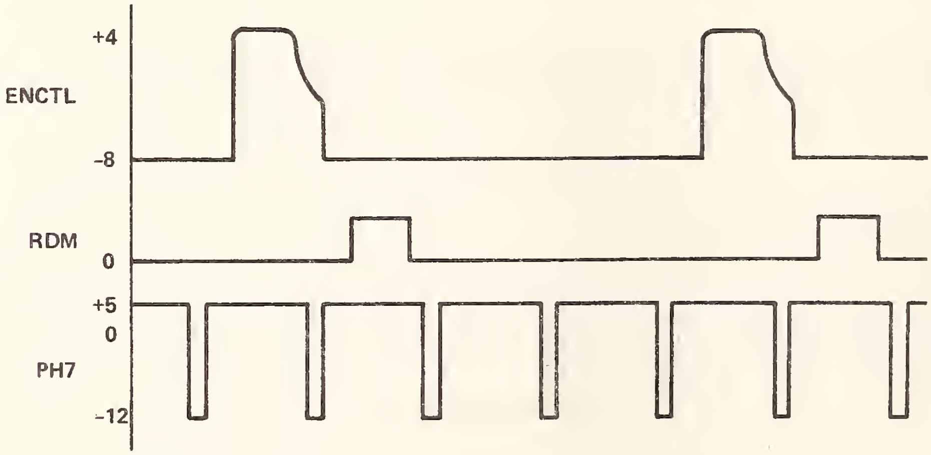
## 8.4 CONTROL FLAGS AND LOGIC

Another convenient test point for system verification is the Read Memory (RDM) flag. This flag is pulsed between T2 and T6 during every fetch operation. If operation of the CROM has been verified, then this flag should come up properly (figure 8–2).



NS00142

*Figure 8–1. IMP-16C Clocks*



NS00143

*Figure 8–2. Microcycle Timing Sequence for JSR Instruction*

The data buses can be checked out by the following:

- Reading in some switch values and seeing if they appear on the bus data output lines.
- Checking the address register to see if the data bus value is being latched during T4 when the RDM flag is up.

## 8.6 DIAGNOSTIC PROGRAMS

The IMP-16C card may be checked out functionally with the use of special diagnostic programs stored in four electrically programmable ROMs (or PROMs). They may be used for system verification by inserting them in the read-only memory sockets in the card and observing the various status indications provided by the program. The diagnostic routines are designed to exercise all parts of the CPU for proper operation. These programs are available as an option (IMP-16F/501).

## 8.7 OPERATING PROCEDURES

For the proper functioning of the IMP-16C in a system environment, the following points should be observed carefully.

(1) If any of the following signal lines to the IMP-16C is not used, it should be tied to a logic "0" (ground) level:

(a) Any unused jump conditions (JC12 through JC15; EXEC)
(b) General interrupt line (INTRA)
(c) Control Panel interrupt line (CPINT)

(2) If the output disable feature of the address bus is not used, the ODIS line should be tied to a logic "0" (ground) level.

(3) An interrupt control input (INTCTL) is provided for externally enabling the interrupt system. For normal operation, this line should be tied to the interrupt enable (INTEN) provided by the CPU. If the system environment is such that the interrupt system is not used at all, the INTCTL input should be tied to a logic "0."

(4) If slow memories are used (access times in excess of 850 ns), additional clock stretching must be provided by the user (see chapter 9).

(5) On the IMP-16C/200 and 300 cards, the circuit that supplies the switched −12 volts (SVGG) is part of the card itself; therefore, no external −12-volt SVGG should be connected to pins 11 and 12 of the card.

# Chapter 9

# USER OPTIONS

Several user-defined options are available on the basic IMP-16C card. These options pertain to system timing, bus selection, and memory management; these are implemented by removable jumper connections on the IMP-16C/200 and IMP-16C/300 cards.

## 9.1 EXTERNAL CLOCK HOLD

An external input (EXHOLD) is provided so the user may supply his own clock hold signal. If this input is used, jumper W6 should be removed. When using this feature, care must be taken to synchronize the external signal with the timing of the processor. This is done by making sure that the generated hold signal comes up in the middle of the time slot preceding the phase being stretched. Similarly, at the end of the hold period, the signal must be released at least 50 ns before the start of the next time slot. See figure 8–1 for this type of timing relationship.

## 9.2 INPUT BUS SELECTION

The input bus structure of the IMP-16C is arranged such that memory data and peripheral data are multiplexed under control of the delayed read-memory flag (RDM–Q1). It is possible to change this multiplexing operation by disabling the RDM–Q1 signal (removing jumper W7) and supplying an externally generated signal on the DSLCT input. The user may wish to do this in case his system requirements call for a common input bus for peripheral devices and any add-on memory.

## 9.3 ADDRESS-BUS DISABLE

The address bus on the IMP-16C (ADX00 through ADX15) may be placed in a TRI-STATE mode by taking the ODIS input to a logic "1" level. This feature allows the forcing of external addresses on the bus without CPU intervention. During normal operation, the ODIS line is at a logic "0" level.

## 9.4 MEMORY MANAGEMENT

As mentioned in chapter 4 (section 4.5), the on-board memory is controlled by address bit 15. If it is desired to change this method of decoding or to add on additional memory, the user may remove jumpers W1, W2, and W3 and supply externally generated decoding via input lines CS0, CS1, and CS2. To illustrate this, two examples are given below.

> **Example 1:** If it is desired to add on more read-write memory to make a total of 4K words in the address space 0–4K, then remove jumper W1 and provide a signal at CS0 that is a logic "0" when (and only when) ADX08–ADX15 are all logic "0." This will enable the first 256-word block of memory that is on the IMP-16C card.
>
> **Example 2:** If, for example, 36K of memory were required, then remove W1, W2, and W3. CS0 would be generated exactly as in example 1. CS1 would have to be a logic "0" when ADX09–ADX15 are all logic "1" with ADX08 a logic "0," and CS2 would have to be a logic "0" when ADX08–ADX15 are all logic "1."

## 9.5 MEMORY ACCESS TIME

The addition of extra memory to the IMP-16C will increase the capacitive loading on the system, so appropriate allowances must be made for access time degradation. The clock-hold circuit on the IMP-16C is designed to permit access times up to 850 ns. For longer access times, the external clock-hold option should be used (see 9.1).

# Appendix A
# SUMMARY OF INSTRUCTIONS

*Table A–1. IMP-16C Basic Instruction Set (Executed by CROM I)*

| Instruction | Mnemonic | Execution Cycles | Memory Read Cycles | Memory Write Cycles |
|---|---|---|---|---|
| **Memory Reference Instructions** | | | | |
| Load | LD | 5 | 2 | – |
| Load Indirect[1] | LD | 5 | 3 | – |
| Store | ST | 6 | 1 | 1 |
| Store Indirect[1] | ST | 8 | 2 | 1 |
| Add | ADD | 5 | 2 | – |
| Subtract | SUB | 5 | 2 | – |
| Jump | JMP | 3 | 1 | – |
| Jump Indirect[1] | JMP | 5 | 2 | – |
| Jump to Subroutine | JSR | 4 | 1 | – |
| Jump to Subroutine Indirect[1] | JSR | 6 | 2 | – |
| Increment and Skip if Zero | ISZ | 7,8 if SKIP | 2 | 1 |
| Decrement and Skip if Zero | DSZ | 8,9 if SKIP | 2 | 1 |
| Skip if AND is Zero | SKAZ | 6,7 if SKIP | 2 | – |
| Skip if Greater | SKG | Like Signs: 8,9 if SKIP<br>Unlike Signs: 9,10 if SKIP | 2 | – |
| Skip if Not Equal | SKNE | 6 | 2 | – |
| And | AND | 5 | 2 | – |
| Or | OR | 5 | 2 | – |
| **Register Reference Instructions** | | | | |
| Push on to Stack Register | PUSH | 3 | 1 | – |
| Pull from Stack | PULL | 3 | 1 | – |
| Add Immediate, Skip if Zero | AISZ | 4,5 if SKIP | 1 | – |
| Load Immediate | LI | 3 | 1 | – |
| Complement and Add Immediate | CAI | 3 | 1 | – |
| Register Copy | RCPY | 6 | 1 | – |
| Exchange Register and Top of Stack | XCHRS | 5 | 1 | – |
| Exchange Registers | RXCH | 8 | 1 | – |
| Register And | RAND | 6 | 1 | – |
| Register Exclusive Or | RXOR | 6 | 1 | – |
| Register Add | RADD | 3 | 1 | – |
| Shift Left | SHL | 4 + 3K | 1 | – |
| Shift Right | SHR | 4 + 3K | 1 | – |
| Rotate Left | ROL | 4 + 3K | 1 | – |
| Rotate Right | ROR | 4 + 3K | 1 | – |

[1] The symbol @ must precede the designation of the memory location whose contents become the effective address by indirection.

*Table A–1. IMP-16C Basic Instruction Set (Continued)*

| Instruction | Mnemonic | Execution Cycles | Memory Read Cycles | Memory Write Cycles |
|---|---|---|---|---|
| **Input/Output, Flag, and Halt Instructions** | | | | |
| Set Flag | SFLG | 4 | 1 | – |
| Pulse Flag | PFLG | 4 | 1 | – |
| Push Flags on Stack | PUSHF | 4 | 1 | – |
| Pull Flags from Stack | PULLF | 5 | 1 | – |
| Register In | RIN | 7 | 1 | – |
| Register Out | ROUT | 7 | 1 | – |
| Halt | HALT | – | – | – |
| **Transfer of Control Instructions** | | | | |
| Branch-On Condition | BOC | 4,5 if branch | 1 | – |
| Return from Subroutine | RTS | 4 | 1 | – |
| Return from Interrupt | RTI | 5 | 1 | – |
| Jump to Subroutine Implied | JSRI | 4 | 1 | – |

$$\text{Execution Time} = (E + 0.25R + 0.25W)\ T$$

where

E = number of execution cycles

R = number of memory read cycles

W = number of memory write cycles

T = time for one microcycle

The 0.25 factor for the read and write cycles is included because the main clocks are stopped for two periods during read and write operations with memory. If the clock-stop feature is not used, then there is no overhead for the read/write operations; hence 0.25 may be replaced by 0. For the shift and rotate instructions, K refers to the number of positions shifted or rotated.

As an example of the use of the formula, let T = 1.4 microseconds; then, a load-instruction execution would take

$$(5 + 0.50)\ 1.4 = 7.7 \text{ microseconds}$$

A store-instruction execution would take

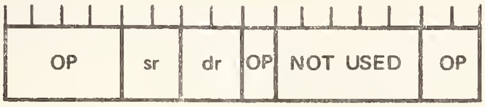
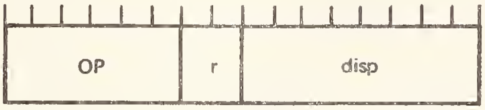
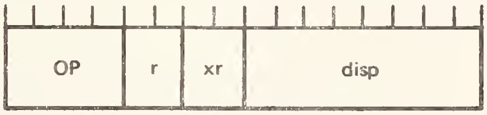
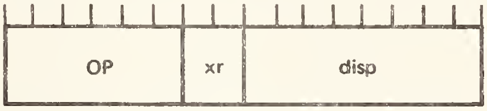
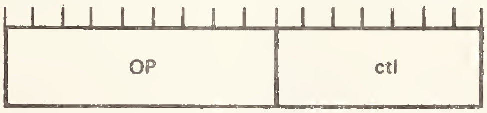
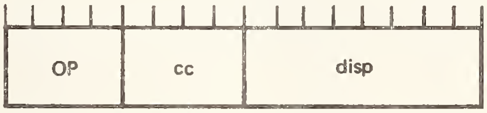
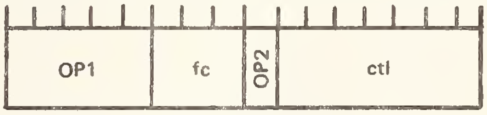
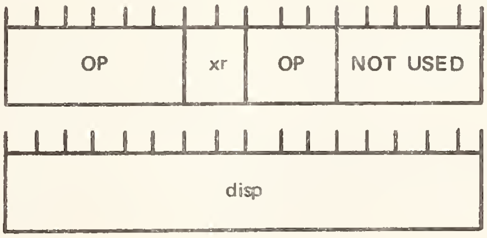
$$(6 + 0.25 + 0.25)\ 1.4 = 9.09 \text{ microseconds}$$

*Table A–2. IMP-16C Extended Instruction Set (Executed by CROM II)*

| Instruction | Mnemonic | Execution Cycles | Memory Read Cycles | Memory Write Cycles |
|---|---|---|---|---|
| Multiply | MPY | 106 to 122 | 3 | – |
| Divide | DIV | 125 to 159 | 3 | – |
| Double Precision Add | DADD | 12 | 4 | – |
| Double Precision Subtract | DSUB | 12 | 4 | – |
| Load Byte | LDB | 20 (left)<br>12 (right) | 4 | – |
| Store Byte | STB | 24 (left)<br>17 (right) | 4 | 1 |
| Set Status Flag | SETST | 17 to 36 | 1 | – |
| Clear Status Flag | CLRST | 17 to 36 | 1 | – |
| Skip If Status Flag True | SKSTF | 19 to 39 | 1 | – |
| Set Bit | SETBIT | 15 to 34 | 1 | – |
| Clear Bit | CLRBIT | 15 to 34 | 1 | – |
| Complement Bit | CMPBIT | 15 to 34 | 1 | – |
| Skip If Bit True | SKBIT | 19 to 39 | 1 | – |
| Interrupt Scan | ISCAN | 9 to 80 | 1 | – |
| Jump Indirect to Level Zero Interrupt | JINT | 7 | 2 | – |
| Jump Through Pointer | JMPP | 7 | 3 | – |
| Jump to Subroutine Through Pointer | JSRP | 8 | 3 | – |

# Appendix B
# FORMAT OF INSTRUCTIONS

A summary of the instruction types and their assembler language formats is given below for reference. A more-detailed breakdown of the instruction codes is shown in the next table; it is suitable for hand-coding small programs.

| Instruction Type | Machine Format | Assembler Language Format | | Remarks |
|---|---|---|---|---|
| Register to Register |  OP<br>sr<br>dr<br>OP<br>NOT USED<br>OP | Op | sr, dr | |
| Register to Memory |  OP<br>r<br>disp | Op | r, disp | |
| Memory Reference (Class 1) |  OP<br>r<br>xr<br>disp | Op<br>Op | r, disp (xr)<br>r, @disp (xr) | Direct<br>Indirect |
| Memory Reference (Class 2) |  OP<br>xr<br>disp | Op<br>Op | disp (xr)<br>@disp (xr) | Direct<br>Indirect |
| I/O and Miscellaneous |  OP<br>ctl | Op | ctl | |
| Branch |  OP<br>cc<br>disp | Op | cc, disp | |
| Control Flags |  OP1<br>fc<br>OP2<br>ctl | Op | fc, ctl | |
| Memory Reference (Double Word) |  OP<br>xr<br>OP<br>NOT USED<br>disp | Op | (xr)<br>disp | |

**Explanation of Symbols**

Op – Instruction Mnemonic

OP – Operation Code

sr – Source Register Value

dr – Destination Register Value

xr – Index Register Value (2 or 3)

disp – Displacement Value

cc – Condition Code Value

r – Register Value

ctl – Control Bits Value

*Table B–1. Basic Instruction Set with Bit Patterns*

| Mnemonic | Base |
|---|---|
| LD | 8000 |
| LD Indirect | 9000 |
| ST | A000 |
| ST Indirect | B000 |
| ADD | C000 |
| SUB | D000 |
| SKG | E000 |
| SKNE | F000 |

**Word Format**

I = BASE ∨ r ∨ xr ∨ disp

| r | REGISTER |
|---|---|
| 0000 | 0 |
| 0400 | 1 |
| 0800 | 2 |
| 0C00 | 3 |

| xr | ADDRESSING TECHNIQUE |
|---|---|
| 0000 | BASE PAGE |
| 0100 | PC RELATIVE |
| 0200 | INDEXED – AC2 |
| 0300 | INDEXED – AC3 |

| | |
|---|---|
| AND | 6000 |
| OR | 6800 |
| SKAZ | 7000 |

| r | REGISTER |
|---|---|
| 0000 | 0 |
| 0400 | 1 |

| | | | |
|---|---|---|---|
| ISZ | 7800 | JMP Indirect | 2400 |
| DSZ | 7C00 | JSR | 2800 |
| JMP | 2000 | JSR Indirect | 2C00 |

**Word Format**

I = BASE ∨ cc ∨ disp

BOC 1000

| Branch on | INTRPT=1 (when enabled) | AC0=0 | AC0 ⩾ 0 | AC0 ODD | AC0 Bit 1=1 | AC0≠0 | CPINT =1 | START =1 |
|---|---|---|---|---|---|---|---|---|
| CC | 0000 | 0100 | 0200 | 0300 | 0400 | 0500 | 0600 | 0700 |

| Branch on | STFL=1 | INTEN=1 | CYOV=1 | AC0⩽0 | USER | USER | USER | USER |
|---|---|---|---|---|---|---|---|---|
| CC | 0800 | 0900 | 0A00 | 0B00 | 0C00 | 0D00 | 0E00 | 0F00 |

**Word Format**

I = BASE ∨ r ∨ disp

| | | |
|---|---|---|
| AISZ | 4800 | |
| LI | 4C00 | |
| CAI | 5000 | |
| PUSH | 4000 | |
| PULL | 4400 | |
| XCHRS | 5400 | |
| ROR/ROL | 5800 | LEFT DISP POSITIVE |
| SHR/SHL | 5C00 | RIGHT DISP NEGATIVE |

| r | REGISTER |
|---|---|
| 0000 | 0 |
| 0100 | 1 |
| 0200 | 2 |
| 0300 | 3 |

**Word Format**

I = BASE ∨ sr ∨ dr

| | |
|---|---|
| RADD | 3000 |
| RXCH | 3080 |
| RCPY | 3081 |
| RXOR | 3082 |
| RAND | 3083 |

| sr | dr | REGISTER |
|---|---|---|
| 0000 | 0000 | 0 |
| 0400 | 0100 | 1 |
| 0800 | 0200 | 2 |
| 0C00 | 0300 | 3 |

*Table B–1. Basic Instruction Set with Bit Patterns (Continued)*

| Mnemonic | Base | fc | FLAG | Word Format |
|---|---|---|---|---|
| SFLG | 0800 | | | I = BASE ∨ fc ∨ ctl |
| PFLG | 0880 | 0000 | 8 | |
| | | 0100 | 9 | |
| | | 0200 | 10 | |
| | | 0300 | 11 | |
| | | 0400 | 12 | |
| | | 0500 | 13 | |
| | | 0600 | 14 | |
| | | 0700 | 15 | |

| | | | | | | Word Format |
|---|---|---|---|---|---|---|
| | | | | | | I = BASE ∨ ctl |
| HALT | 0000 | RTI | 0100 | RIN | 0400 | |
| PUSHF | 0080 | RTS | 0200 | ROUT | 0600 | |
| PULLF | 0280 | JSRI | 0380 | (Address range = FF80 to FFFF) | | |

The instruction is formed by the inclusive Or of each field. For example, the instruction RADD 2,3 is coded as X'3B00

For instructions that use the CTL field, only the first 7 bits (bits 0 through 6) are considered.

Examples of coding follow:

**Example 1**

RADD 2,3

```
       BASE = 3000
         sr = 0800
         dr = 0300
INSTRUCTION = 3B00
```

**Comments**

Add AC2 to AC3.

**Example 2**

JMP–1 (3)

```
       BASE = 2000
         xr = 0300
       disp = 00FF
INSTRUCTION = 23FF
```

**Comments**

Jump to the location specified by the index register AC3 modified by the displacement –1.

**Example 3**

SHR 0,1

```
       BASE = 5C00
          r = 0000
       disp = 00FF
INSTRUCTION = 5CFF
```

**Comments**

Shift the contents of AC0 one place to the right.

*Table B–2. Extended Instruction Set with Bit Patterns*

| Mnemonic | Base | Word Format |
|---|---|---|
| MPY | 0480 | I (Word 1) = BASE ∨ xr |
| DIV | 0490 | I (Word 2) = disp |
| DADD | 04A0 | |
| DSUB | 04B0 | |

| xr | ADDRESSING TECHNIQUE |
|---|---|
| 0000 | Direct |
| 0100 | PC Relative |
| 0200 | Indexed – AC2 |
| 0300 | Indexed – AC3 |

| Mnemonic | Base | Word Format |
|---|---|---|
| LDB | 04C0 | I (Word 1) = BASE ∨ xr |
| STB | 04D0 | I (Word 2) = 2. disp ∨ Byte |
| | | Byte = 0 for right, 1 for left. |
| ISCAN | 0510 | |

| Mnemonic | Base | Word Format |
|---|---|---|
| SETST | 0700 | I = BASE ∨ Bit |
| CLRST | 0710 | |
| SETBIT | 0720 | |
| CLRBIT | 0730 | |
| SKSTF | 0740 | |
| SKBIT | 0750 | |
| CMPBIT | 0760 | |

| Mnemonic | Base | BASADR | MAXADR | Word Format |
|---|---|---|---|---|
| JSRP | 0300 | (0100) | 007F | I = BASE ∨ Incr |
| JMPP | 0500 | (0100) | 000F | Incr = ADDR – BASADR |
| JINT | 0520 | 0120 | 000F | BASADR ⩽ ADDR ⩽ BASADR + MAXADR |

**Example 1**

LDB X'A0A

BASE = 04C0
xr = 0000
INSTRUCTION = 04C0 (Word 1)

disp = X'0A0A
Byte = 0 (Right)

INSTRUCTION = 1413 (Word 2)

**Comments**

Load AC0 with the right byte of location X'0A0A; direct addressing.

| Example 2 | Comments |
|---|---|
| JMPP 2 | Jump through pointer located at location X'0102. |

```
       BASE = 0500
     BASADR = (0100) Implied
       INCR = 0002

INSTRUCTION = 0502
```

# Appendix C
# MEMORY ARRANGEMENT

The arrangement of memory in this system is shown in figure C–1. The entire base page consists of read/write memory; that is, locations $(0000)_{16}$ to $(00FF)_{16}$. The top sector of memory $(FE00)_{16}$ to $(FFFF)_{16}$ is implemented with read-only memory. The read-only memory on the board is enabled whenever bit 15 of the address is 1.

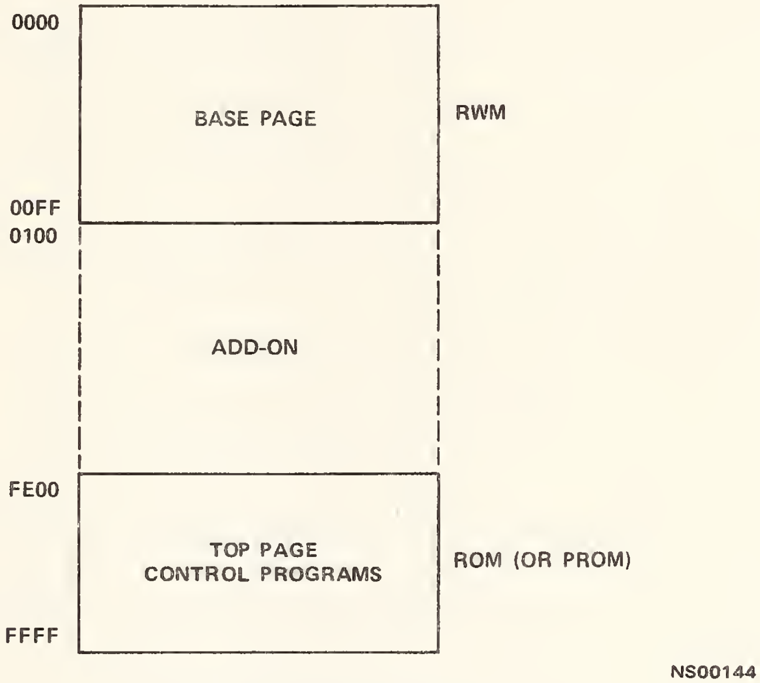


*Figure C–1. Memory Layout*

The memory arrangement shown above may be moved around to suit the convenience of the user. Jumper connections are provided on the IMP-16C card (W1, W2, and W3) to disable the on-board address decode logic. By removing these and using the external chip select inputs (CS0, CS1, and CS2), the user may generate his own decode signals to move the memory pages wherever desired.

# Appendix D
# IMP-16C NOMENCLATURE

*Table D–1. Nomenclature Used in Circuit Schematics and Text*

| Signal Name | Description |
|---|---|
| ADEN | Address Enable Signal |
| ADX (0), (1), ..., (15) | Address Lines to Memory |
| ALU | Arithmetic/Logic Unit |
| BDO (0), (1), ..., (15) | Buffered Data-Out Lines |
| CI | Memory Cycle Initiate |
| CJMUX | Conditional Jump Multiplexer |
| CLK, CLK* | Master Clock Signals |
| CPINP | Control Panel Interrupt Acknowledge |
| CPINT | Control Panel Interrupt |
| CROM | Control Read-Only Memory |
| CSH0, CSH3 | Carry/Shift Signal Lines (RALU) |
| CS0, CS1, CS2 | Memory Chip-Select Lines |
| CYOV | Carry or Overflow Signal |
| C81, C23, C45, C67 | Clock Signals, each lasting for two periods |
| DATA (0), (1), (2), and (3) | Data Bus Lines |
| DI (0), (1), (2), and (3) | Data Input Lines to the CROM |
| DISTR | Data Input Strobe |
| ENCTL | Enable Control Signal |
| START | Start or continue signal to restart operation |
| FLAG0, FLAG12 | User status flags |
| F8, F11, ..., F15 | User control flags |
| HCLK | Clock Hold Signal |
| HLT | Halt Flag |
| HOCSH | High-Order Carry/Shift Signal Line (CROM) |
| HOLD | Phase 4 Hold Signal |
| INIT* | Initialize Line (Complemented) |
| INTEN | Interrupt Enable Flag |
| INTRPT | Delayed Interrupt Signal |
| INTRA | Interrupt Request Signal |
| INTCTL | Interrupt Control Signal |
| JCSTR | Jump Condition Strobe Signal |
| JFA | Jump Flag Addresses |

*Table D–1. Nomenclature Used in Circuit Schematics and Text (Continued)*

| Signal Name | Description |
|---|---|
| LDAR | Load Address Register Flag |
| LOCSH | Low-Order Carry/Shift Signal Line (CROM) |
| MDO (0), (1), . . ., (15) | Memory Data Out |
| MUX | Multiplexer |
| NCB (0), (1), (2), and (3) | Complemented Control Bus Lines |
| NFLEN | Flag Enable Signal Line (Complemented) |
| NJCOND | Jump Condition Input Line (Complemented) |
| NREQ0 | Register Equal Zero Signal (Complemented) |
| POC | Power-On Condition |
| PROM | Electrically Programmed Read-Only Memory |
| RALU | Register/Arithmetic/Logic Unit |
| RDM | Read Memory Flag |
| RDM–Q1 | Delayed Read Memory Flag |
| RDP | Read Peripheral Flag |
| RFREQ | Memory Refresh Request Signal |
| RFSH | Memory Refresh Initiate |
| SEL | Select Flag |
| SININ | Sign-In Signal Line |
| STF | Stack Full Signal Line |
| STFL | Stack Full Interrupt Signal |
| SVRST | Save/Restore Flag |
| SW | Switch Data (Input Port for Peripheral Data) |
| SYSCLR* | System Clear (Complemented) |
| $V_{GG}$, $V_{SS}$ | Supply Voltages |
| WRM | Write Memory Flag |
| WRMP | Write Memory Pulse |
| WRP | Write Peripheral Flag |
| WRPA | Write Peripheral Strobe A |
| WRPB | Write Peripheral Strobe B |
| WRP3 | Write Peripheral Strobe |
| PH1, PH3, PH5, and PH7 | Clock Phase Times 1, 3, 5, and 7. (Each of these phase times corresponds to a clock pulse: T1, T3, T5, and T7.) |

NOTE: An asterisk (*) after a signal name (except a CROM or a RALU signal) denotes a complemented signal. Complementation is denoted for CROM and RALU signals by the prefix N as part of the signal names.

# Appendix E
# LIST OF PIN CONNECTIONS AND SIGNALS ON IMP-16C CARD

*Table E–1. IMP-16C Pin Numbers and Corresponding Signal Names*

| Pin Number | Signal Name | Pin Number | Signal Name |
|---|---|---|---|
| 1 | Ground | 2 | Ground |
| 3 | Ground | 4 | Ground |
| 5 | +5 volts | 6 | +5 volts |
| 7 | +5 volts | 8 | +5 volts |
| 9 | FLAG12 – Status Flag from RALU | 10 | ADX00 – Address Line, Bit 00 |
| 11 | SVGG – Switched –12 volts[1] | 12 | SVGG – Switched –12 volts[1] |
| 13 | –10 volts for Read/Write Memory (MM1101A2) | 14 | –9 volts for Read/Write Memory (MM1101A2) |
| 15 | INTRA – Interrupt Request Signal | 16 | INIT* – Initialize (Complemented) |
| 17 | Not Used | 18 | ADX07 – Address Line, Bit 07 |
| 19 | Not Used | 20 | ADX06 – Address Line, Bit 06 |
| 21 | CS0 – Memory Chip-Select Line | 22 | FLAG0 – Status Flag from RALU |
| 23 | ADX02 – Address Line, Bit 02 | 24 | ADX01 – Address Line, Bit 01 |
| 25 | ADX03 – Address Line, Bit 03 | 26 | Not Used |
| 27 | BDO08 – Buffered Data Out, Bit 08 | 28 | ADX05 – Address Line, Bit 05 |
| 29 | ADX04 – Address Line, Bit 04 | 30 | C3B* – Timing Signal |
| 31 | –12 volts | 32 | –12 volts |
| 33 | Not Used | 34 | CS1 – Memory Chip-Select Line |
| 35 | CS2 – Memory Chip-Select Line | 36 | ODIS – Address Bus Disable |
| 37 | Not Used | 38 | Not Used |
| 39 | Not Used | 40 | MDO00 – Memory Data Out, Bit 00 |
| 41 | MDO01 – Memory Data Out, Bit 01 | 42 | Not Used |
| 43 | MDO02 – Memory Data Out, Bit 02 | 44 | MDO03 – Memory Data Out, Bit 03 |
| 45 | Not Used | 46 | Not Used |
| 47 | MDO04 – Memory Data Out, Bit 04 | 48 | MDO05 – Memory Data Out, Bit 05 |
| 49 | Not Used | 50 | MDO06 – Memory Data Out, Bit 06 |
| 51 | Not Used | 52 | MDO07 – Memory Data Out, Bit 07 |
| 53 | WRPB – Write Peripheral Strobe B | 54 | Not Used |
| 55 | Not Used | 56 | BDO05 – Buffered Data Out, Bit 05 |
| 57 | DISTR*– Data Input Strobe (Complemented) | 58 | BDO01 – Buffered Data Out, Bit 01 |
| 59 | BDO04 – Buffered Data Out, Bit 04 | 60 | BDO00 – Buffered Data Out, Bit 00 |
| 61 | BDO03 – Buffered Data Out, Bit 03 | 62 | MDO14 – Memory Data Out, Bit 14 |
| 63 | BDO07 – Buffered Data Out, Bit 07 | 64 | BDO02 – Buffered Data Out, Bit 02 |
| 65 | BDO06 – Buffered Data Out, Bit 06 | 66 | WRPA – Write Peripheral Strobe A |
| 67 | BDO09 – Buffered Data Out, Bit 09 | 68 | BDO11 – Buffered Data Out, Bit 11 |
| 69 | BDO14 – Buffered Data Out, Bit 14 | 70 | BDO15 – Buffered Data Out, Bit 15 |

[1] - Should not be used on IMP-16/200 and 300 cards.

*Table E–1. IMP-16C Pin Numbers and Corresponding Signal Names (Continued)*

| Pin Number | Signal Name | Pin Number | Signal Name |
|---|---|---|---|
| 71 | Ground | 72 | Ground |
| 73 | BDO10 – Buffered Data Out, Bit 10 | 74 | BDO13 – Buffered Data Out, Bit 13 |
| 75 | BDO12 – Buffered Data Out, Bit 12 | 76 | MDO10 – Memory Data Out, Bit 10 |
| 77 | MDO13 – Memory Data Out, Bit 13 | 78 | MDO12 – Memory Data Out, Bit 12 |
| 79 | SW00 – Switch Data, Bit 00 | 80 | DSLCT – Input Data Select |
| 81 | SW02 – Switch Data, Bit 02 | 82 | MDO08 – Memory Data Out, Bit 08 |
| 83 | Not Used | 84 | SW01 – Switch Data, Bit 01 |
| 85 | INTCTL – Interrupt Control | 86 | SW03 – Switch Data, Bit 03 |
| 87 | MDO09 – Memory Data Out, Bit 09 | 88 | MDO11 – Memory Data Out, Bit 11 |
| 89 | WRM – Write Memory Flag | 90 | SW08 – Switch Data, Bit 08 |
| 91 | SW10 – Switch Data, Bit 10 | 92 | SW09 – Switch Data, Bit 09 |
| 93 | SW11 – Switch Data, Bit 11 | 94 | CLK81 – Timing Signal |
| 95 | SW12 – Switch Data, Bit 12 | 96 | SW13 – Switch Data, Bit 13 |
| 97 | SW14 – Switch Data, Bit 14 | 98 | SW04 – Switch Data, Bit 04 |
| 99 | C45 – Timing Signal | 100 | SW06 – Switch Data, Bit 06 |
| 101 | ADX13 – Address Line, Bit 13 | 102 | EXHOLD – External Hold |
| 103 | SW05 – Switch Data, Bit 05 | 104 | JC14 – General-Purpose Jump Condition |
| 105 | SW07 – Switch Data, Bit 07 | 106 | SW15 – Switch Data, Bit 15 |
| 107 | JC15 – General-Purpose Jump Condition | 108 | MDO15 – Memory Data Out, Bit 15 |
| 109 | ADX11 – Address Line, Bit 11 | 110 | ADX10 – Address Line, Bit 10 |
| 111 | ADX08 – Address Line, Bit 08 | 112 | ADX09 – Address Line, Bit 09 |
| 113 | RDM – Read Memory Flag | 114 | ADX12 – Address Line, Bit 12 |
| 115 | ADX14 – Address Line, Bit 14 | 116 | ADX15 – Address Line, Bit 15 |
| 117 | CPINP – Control Panel Interrupt | 118 | CPINT – Control Panel Interrupt |
| 119 | JC13 – General-Purpose Jump Condition | 120 | WRMP – Write Memory Pulse |
| 121 | RDP – Read Peripheral Flag | 122 | WRP – Write Peripheral Flag |
| 123 | START – Start Signal | 124 | RFREQ – Memory Refresh Request |
| 125 | HLT* (FLAG) | 126 | SYSCLR* – System Clear (Complemented) |
| 127 | CLK* – Master Clock (Complemented) | 128 | CLK – Master Clock |
| 129 | JC12 – General-Purpose Jump Condition | 130 | F8 – General-Purpose User Flag |
| 131 | INT EN – Interrupt Enable Flag | 132 | F15 – General-Purpose User Flag |
| 133 | F11 – General-Purpose User Flag | 134 | F14 – General-Purpose User Flag |
| 135 | F13 – General-Purpose User Flag | 136 | F12 – General-Purpose User Flag |
| 137 | +5 volts | 138 | +5 volts |
| 139 | +5 volts | 140 | +5 volts |
| 141 | Ground | 142 | Ground |
| 143 | Ground | 144 | Ground |

NOTE: 1. Odd-numbered pins are on component side and even-numbered pins are on solder side of IMP-16C card.

2. Pins 83 and 55 are used for CYCLE INITIATE and EXTERNAL REFRESH functions, respectively, in the IMP-16P system (IMP-16P/200, /300).

SUPPLEMENT 1 TO PUB. NO. 4200021C

Integrated MicroProcessor-16C

# IMP-16C

# INTERFACING GUIDE

January 1974

© National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, California 95051

## PREFACE

This application note supplements the IMP-16C Application Manual and supplies information pertaining to the implementation of peripheral interfacing for the IMP-16C. The user should be familiar with the contents of the IMP-16C Application Manual, particularly with respect to the macroinstruction and timing-signal mnemonics.

The material supplied in this application note is for information purposes only and is subject to change without notice. This applies particularly to the circuit diagrams and the computer program listings.

## CONTENTS

Chapter Page

CONTENTS (Continued)

## ILLUSTRATIONS

## TABLES

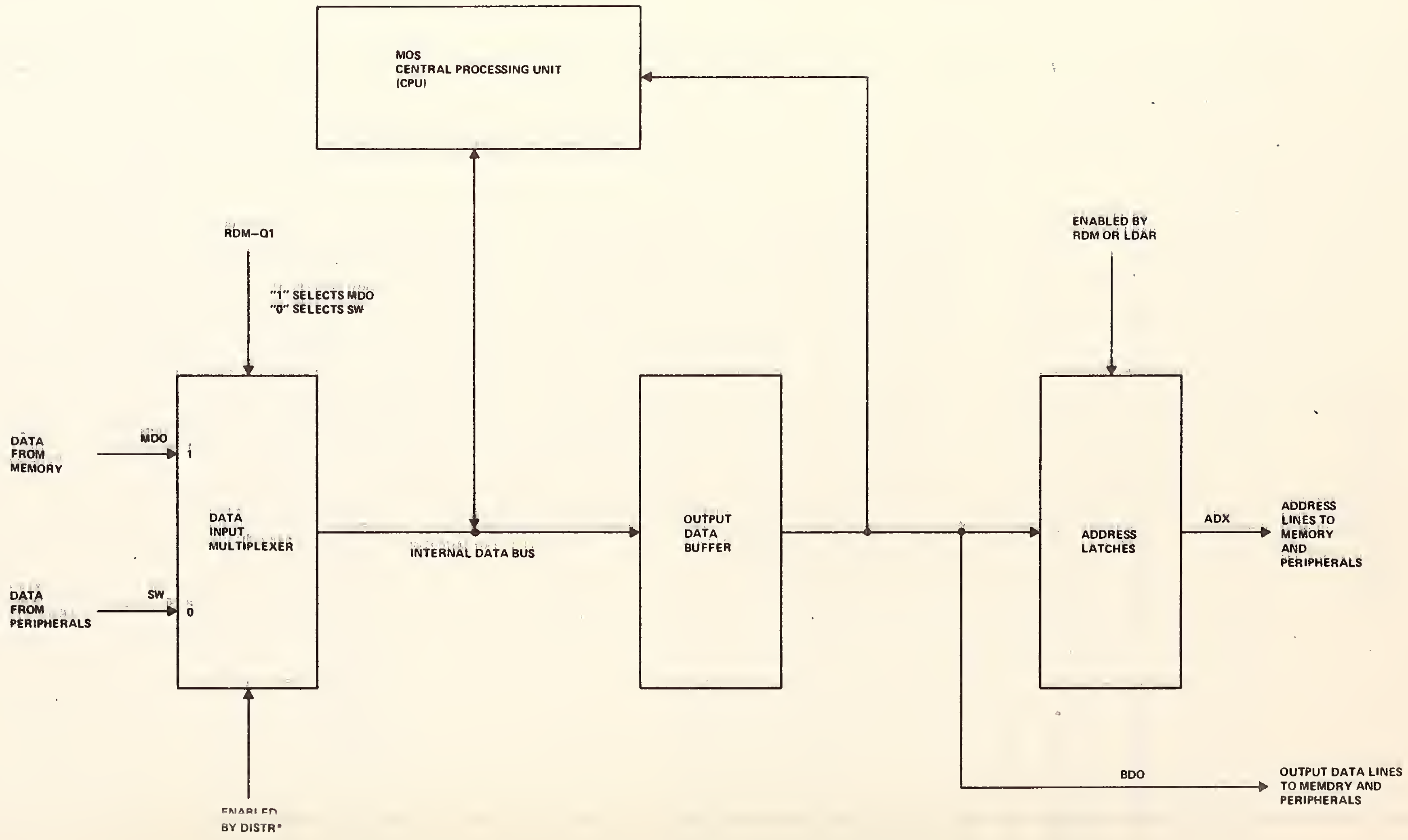


Figure 1-1. IMP-16C Bus Interface Structure

# CHAPTER 1

# GENERAL INFORMATION

This chapter discusses IMP-16C interface considerations, advisability of having a control panel for use with the IMP-16C, and interface methods and programs. More-detailed information on these subjects is presented in the remaining chapters.

## 1.1 IMP-16C INTERFACE CONSIDERATIONS

Peripheral devices communicate with the IMP-16C processor via the SW input bus and the BDO data out bus. A number of user control flags are available to control various I/O operations. In this application note, emphasis is placed on program-controlled operations to illustrate the simplicity of the interface requirements.

When designing a peripheral device interface, a wide range of possibilities exist between a single-purpose (economical) interface and a multi-purpose (expensive) interface. The single-purpose interface must rely on processor control to generate the necessary timing sequences, while the multi-purpose interface requires only a minimal amount of processor control. The trade-off between expense and hardware volume can only be determined by the user's needs. There is nothing inherent in the design of the IMP-16C bus system or the processor that requires a complex interface.

The teletype interface using control flags (described in chapter 3 of this note) is an example of a relatively simple peripheral interface that is flexible enough for all operations but requires a large degree of program control. This is acceptable for most applications where the processor is not doing other functions during teletype operations.

Figure 1-1 is a simplified block diagram of the I/O bus structure of the IMP-16C microprocessor. The timing of the various signals and data lines is given in figure 1-2. Data bits presented on the SW bus are accepted during T7 as determined by the DISTR* pulse. Output data is valid during T4. Input jump conditions and output control flags are valid at the start of T2. Control flags are reset at the beginning of T6. Addresses on the ADX bus start coming up during T3 and are valid at the start of T4.

When interfacing external devices with the IMP-16C, peripheral device addresses may be distinguished from memory addresses by the presence of the RDP or WRP flags that are pulsed during RIN and ROUT instructions, respectively.

Timing for RIN and ROUT instructions is shown in figure 1-3. Each of these instructions takes seven microcycles to execute. During the first microcycle (FETCH), the RDM flag enables the establishment of a memory address on the ADX lines to permit the fetching of the instruction. The next six microcycles effect the execution of the

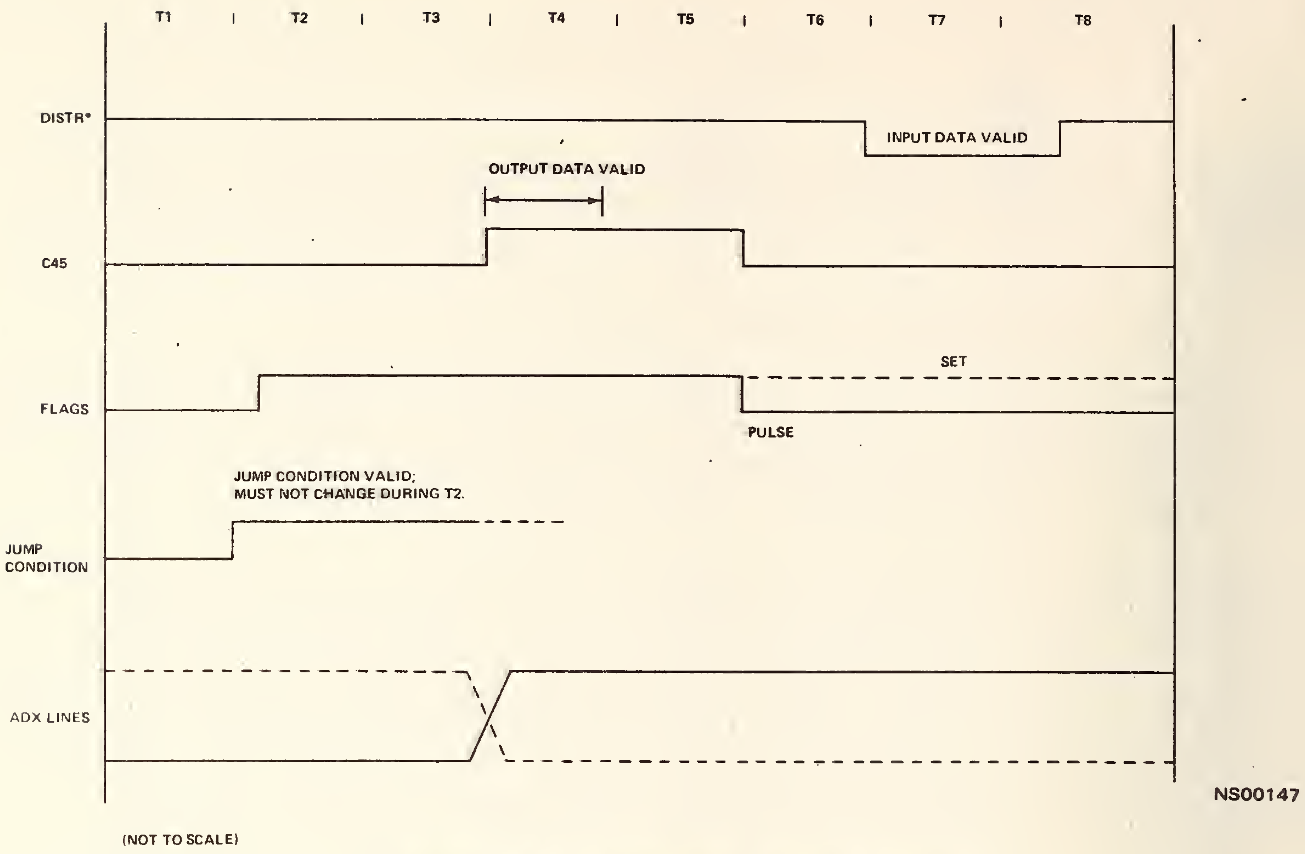


Figure 1-2. IMP-16C Timing Chart

instruction. In the sixth microcycle, the LDAR flag is pulsed, thus enabling the latching of the peripheral address on the ADX lines. These latched addresses are valid through the next microcycle, at which time the RDP or WRP flag is pulsed and the actual data transfer takes place.

During a ROUT instruction, the data present on the BDO lines at T4 of the seventh microcycle is the output data from AC0. During a RIN instruction, the data present on the SW lines at T7 of the seventh microcycle is the input data accepted by the processor and loaded into AC0.

## 1.2 IMP-16C CONTROL PANEL

The IMP-16C is self-contained for control operations; however, for some general-purpose applications a control panel may be needed or convenient to have for debugging programs, entering or retrieving data from the IMP-16C, verifying system operation, troubleshooting, and other operator-controlled purposes.

The control panel described here is one that is relatively uncomplicated and has a minimum number of components. A software service routine, resident in main memory, scans the panel switches and implements appropriate service actions according to switch positions. This control panel and its operations are described in chapter 2.

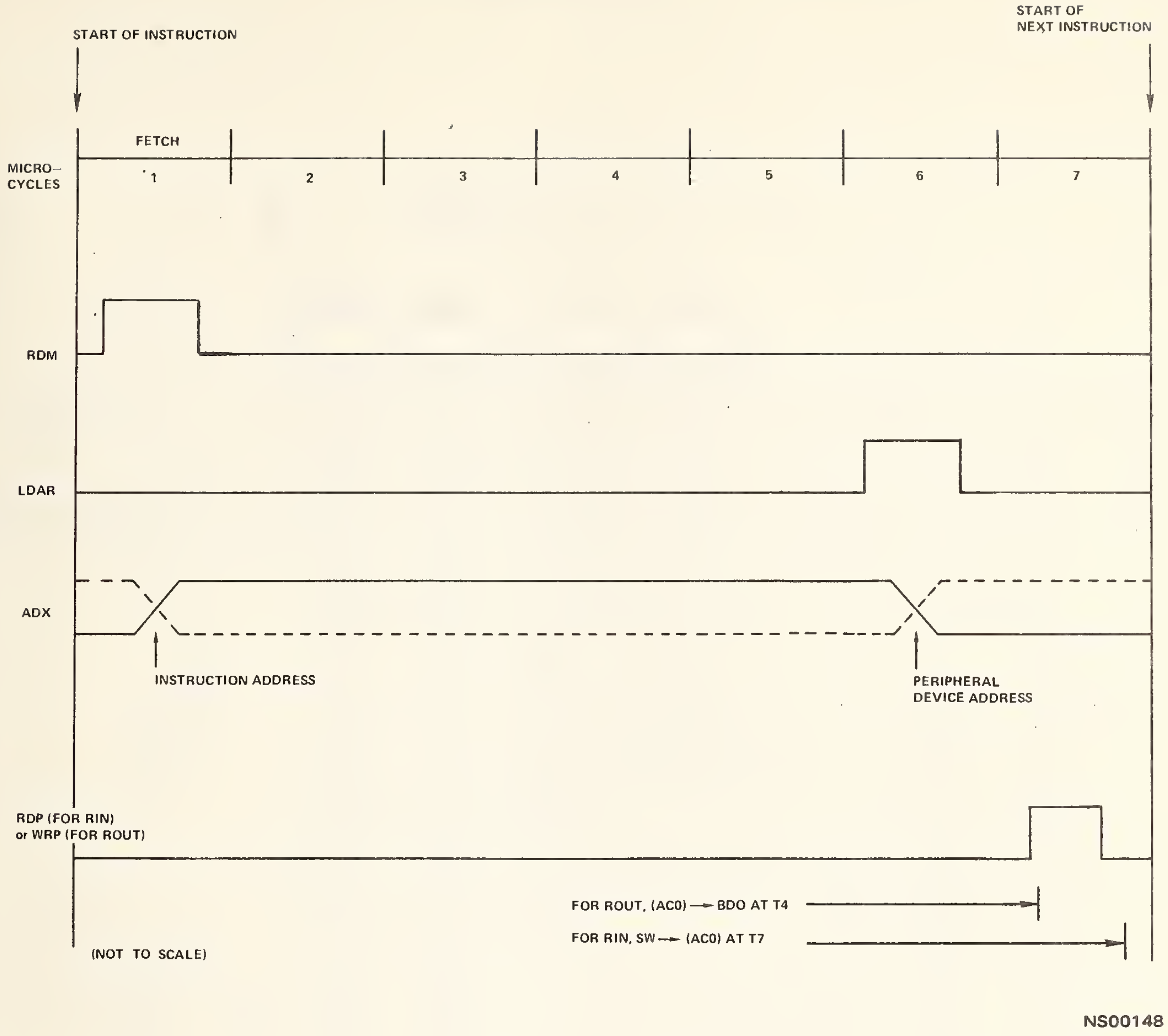
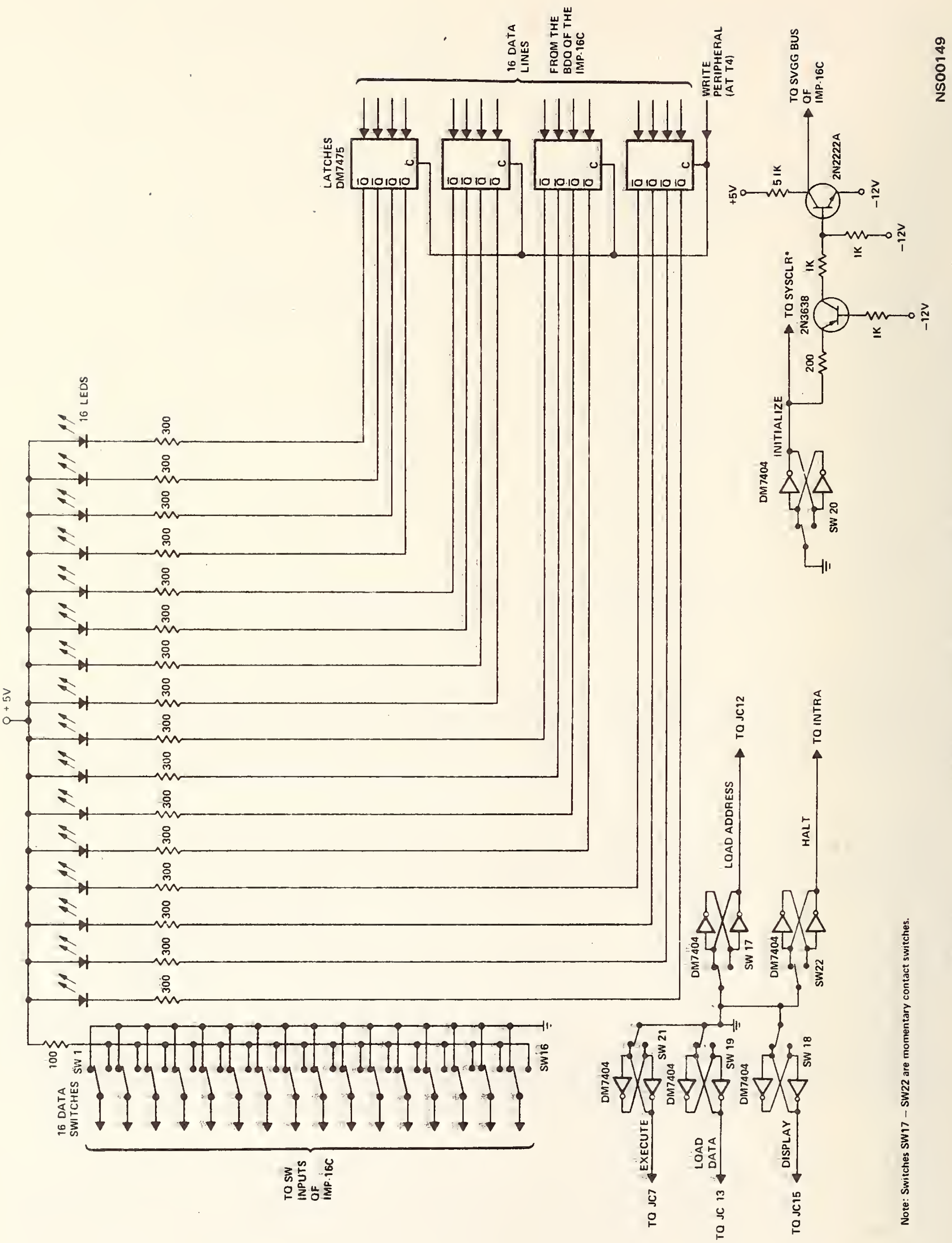

START OF INSTRUCTION
START OF NEXT INSTRUCTION
FETCH
MICRO-CYCLES
1
2
3
4
5
6
7
RDM
LDAR
ADX
INSTRUCTION ADDRESS
PERIPHERAL DEVICE ADDRESS
RDP (FOR RIN) or WRP (FOR ROUT)
FOR ROUT, (AC0) → BDO AT T4
FOR RIN, SW → (AC0) AT T7
(NOT TO SCALE)
NS00148

Figure 1-3. Timing Sequence for RIN and ROUT Instructions

## 1.3 INTERFACE METHODS AND PROGRAMS

Chapters 3 through 7 of this note present several interface methods and related programming considerations as well as a description of interrupt handling. Included are various sample programs that facilitate generating and loading program tapes.

A complete firmware package is available as a PROM and may be run on the IMP-16C (interfaced to a control panel). The listing of this package is given in appendix A.



Note: Switches SW17 – SW22 are momentary contact switches.

Figure 2-1. Control Panel Simplified Schematic Diagram

# CHAPTER 2

# CONTROL PANEL OPERATIONS

## 2.1 GENERAL INFORMATION

The control panel described here is a very simple means of entering and retrieving information from the IMP-16C card. It does not decode any device addresses; it works off the RDP and WRP flags pulsed by the RIN and ROUT instructions. The schematic diagram in figure 2-1 depicts the arrangement of the panel logic. With this panel, six momentary contact switches may be serviced by a software routine. The routine provided with this control panel assumes the following assignments.

| | SWITCH | CONNECTED TO | FUNCTION |
|---|---|---|---|
| | SW21 | JUMP CONDITION 7 | EXECUTE(START) |
| | SW19 | JUMP CONDITION 13 | LOAD DATA |
| | SW18 | JUMP CONDITION 15 | DISPLAY |
| | SW17 | JUMP CONDITION 12 | LOAD ADDRESS |
| | SW22 | INTRA | HALT (interrupt serviced) |
| Provides overriding initialization of CPU | SW20 | SYSCLR* | INITIALIZE |

The physical location of the various switches on the control panel board is shown in figure 2-2; the next few paragraphs describe operating procedures for this panel when it is used with the service program.

Figure 2-1 is a simplified version of the actual control panel logic schematic diagram (provided as part of the control panel kit IMP-16C/882, formerly CTLPLKIT). The switch numbers in the table above refer to switch positions called out on the original diagram.

## 2.2 OPERATING PROCEDURES

The following paragraphs provide the procedures for performing the various operations at the control panel.

### 2.2.1 Loading Main Memory

1. Set the Data Switches to the address of the desired starting location. (Note: locations 0 through 6 in memory are reserved for control program usage.)

2. Press LOAD ADDRESS. The selected address will be displayed on the Bit Display Lights.

3. Set the Data Switches to the data value to be loaded.

4. Press LOAD DATA. The value will be displayed, and the memory-location address will be incremented automatically.

5. Subsequent memory locations may be loaded in consecutive order by setting the desired bit pattern of the value to be loaded and pressing LOAD DATA after each setting.

### 2.2.2 Altering Memory Locations

1. Set the address of the location whose contents are to be altered on the Data Switches, and press LOAD ADDRESS.

2. Set the new value of the data on the Data Switches, and press LOAD DATA.

### 2.2.3 Examining Memory Locations

1. Set the address of the location whose contents are to be displayed on the Data Switches, and press LOAD ADDRESS. The selected address will be displayed on the Bit Display Lights.

2. Press DISPLAY. The contents of the selected location will be displayed, and the memory-location address will be incremented automatically.

3. Subsequent memory locations may be displayed in consecutive order by repeatedly pressing DISPLAY.

### 2.2.4 Loading The Accumulators

1. Set the Data Switches to the number of the accumulator to be loaded (X'0000 for AC0, X'0001 for AC1, and so on), and press LOAD ADDRESS.

2. Set the Data Switches to the value to be loaded, and press LOAD DATA.

3. Repeat steps 1 and 2 for each accumulator to be loaded.

### 2.2.5 Displaying Accumulator Values

1. Set the Data Switches to the number of the accumulator whose contents are to be displayed (similar to step 1 above), and press LOAD ADDRESS.

2. Press DISPLAY.

3. Repeat steps 1 and 2 for each accumulator whose contents are to be displayed.

### 2.2.6 HALT and Continue Operations

The panel HALT switch interrupts the processor and sets it in a HALT mode. If the Data Switches are set to X'0000 and EXECUTE is pressed, then the processor returns control to the control panel routine. Otherwise, an EXECUTE command causes the program to resume where it left off at the time of the HALT.

At the time of a HALT, the return address of the interrupted program is displayed on the Bit Display Lights. This can be noted and used as a resumption address if the normal return from interrupt is not used.

### 2.2.7 Executing A Program

1. Select the starting address of the program on the Data Switches.

2. Press EXECUTE.

After a program has been run, its results may be observed (looking at register values and examining specific memory locations) by coding a JMP X'00 instruction at the last executable instruction; that is, after entering a program, code X'2000 as the final instruction. This returns control to the control panel routine without altering the status of the registers. If register values need not be saved, the same effect can be achieved by pressing INIT.

### 2.2.8 Initialization

The function of the INIT switch is to clear the CPU and return control to the starting sequence. Main memory is not affected by this operation because it has separate power lines. Therefore, a program once loaded into the read/write memory remains unaltered until it is overwritten or power to the entire unit is shut off.

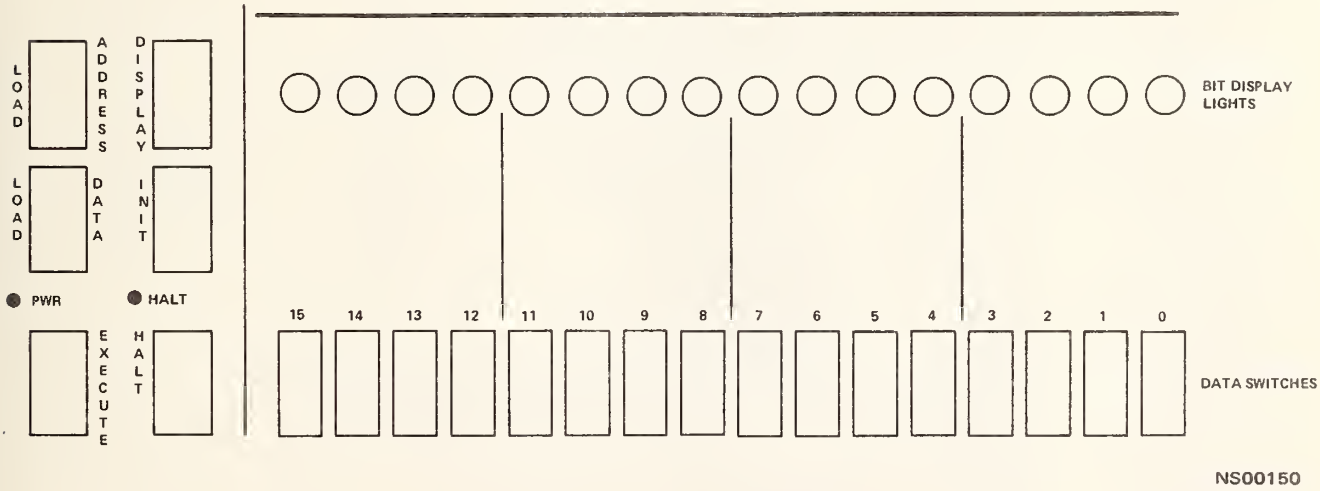


Figure 2-2. Control Panel Arrangement

## 2.3 CONTROL PANEL SERVICE ROUTINE

The program that effects the operations detailed in 2.2.1 through 2.2.7 works on a continuous scan of the front panel active switches. When the processor is powered up, it begins to execute instructions at memory location $FFFE_{16}$ (located in top page in a PROM/ROM). These instructions direct the processor into a 5-cycle wait loop that awaits a front panel command.

The front panel HALT function interrupts normal operation of the processor and sends control to an interrupt service routine. Program listings for the control panel service routine and the interrupt service routine are given on the following pages. It must be noted that this simple control panel by itself is not intended to be a software debug aid, but rather it is a means of manual entry of information into the IMP-16C. As such, when using any of the routines described in this application note, the directions supplied must be followed closely.

NOTE

> All numbers that represent data are written in hexadecimal format. The notation for this format is either the subscript 16 following the number or the prefix X' preceding the number. Example: $4A08_{16}$ means the same as X'4A08.

```
FF86                     .PAGE          'CONTROL PANEL SERVICE ROUTINE'
FF86             ;
FF86             ;                      VERSION 2, APRIL 25, 1973
FF86  FFAD A             .=X'FFAD
FFAD 21B4  A  JSTRT:     .WORD X'21B4
FFAE 21ED  A  JINTR:     .WORD X'21ED
FFAF 0005  A  FIVE:      .WORD   5
FFB0 0001  A  ONE:       .WORD   1
FFB1 8DFC  A  BEGIN:     LD 3, JINTR
FFB2 AC01  A             ST 3,X'01;     LOAD LOCATION 1 WITH JUMP TO INTERRUPT
FFB3 8DF9  A             LD      3,JSTRT
FFB4 AC00  A             ST 3,X'00;     LOAD LOCATION 0 WITH JUMP TO CONTROL PANEL
FFB5 2929  A  START:     JSR SAVE       ; SAVE ACCUMULATORS.
FFB6 0900  A  SET:       SFLG  1        ; ENABLE INTERRUPT SYSTEM.
FFB7 0600  A  ROUT:      ROUT  0
FFB8 1C04  A  WAIT:      BOC C12,LA     ;   'LOAD ADDRESS' SWITCH.
FFB9 1D0A  A             BOC C13,LD     ;   'LOAD DATA' SWITCH.
FFBA 1710  A             BOC C7, EX     ;   'EXECUTE' SWITCH.
FFBB 1F1F  A             BOC C15, DISP  ;   'DISPLAY' SWITCH.
FFBC 21FB  A             JMP .-4        ;   RETURN TO WAIT LOOP.
FFBD 1CFF  A  LA:        BOC C12,LA     ;   CHECK RELEASE.
FFBE 0400  A             RIN 0          ;   READ SWR.
FFBF 3281  A             RCPY 0,2       ;   SAVE ADDRESS IN AC2.
FFC0 3381  A             RCPY 0,3
FFC1 E1ED  A             SKG 0,FIVE;    PREVENTS LOADING OF RESERVED LOCATIONS
FFC2 120D  A             BOC C2, RSRVE
FFC3 21F3  A             JMP ROUT
FFC4 1DFF  A  LD:        BOC C13,LD;    CHECK RELEASE FOR LOAD DATA SWITCH
FFC5 0400  A             RIN 0;         READ SWITCHES
FFC6 F938  A             SKNE 2,LAST6;    PREVENTS LOADING LOCATION 6.
FFC7 21F0  A             JMP WAIT
FFC8 A200  A             ST  0,(2)      ;   LOAD MEMORY
```

```
FFC9 C9E6  A           ADD 2, ONE;    INCREMENT ADDRESS
FFCA 21EC  A           JMP ROUT
FFCB 17FF  A  EX:      BOC C7, EX     ;  CHECK RELEASE
FFCC 0400  A           RIN 0
FFCD 4000  A           PUSH 0         ;  SAVE JUMP ADDRESS IN STACK.
FFCE 2918  A           JSR RSTOR
FFCF 0200  A           RTS 0          ;  FAKING AN INDIRECT JUMP.
FFD0 0600  A  RSRVE:   ROUT 0
FFD1 1D05  A           BOC C13, LDAC
FFD2 1F01  A           BOC C15,DISPAC
FFD3 21FD  A           JMP .-2
FFD4 1FFF  A  DISPAC:  BOC C15,DISPAC;  DISPLAY ACCUMULATOR ROUTINE
FFD5 8302  A           LD 0,2(3)
FFD6 21E0  A           JMP ROUT
FFD7 1DFF  A  LDAC:    BOC C13,LDAC
FFD8 0400  A           RIN 0
FFD9 A202  A           ST 0,2(2)
FFDA 21DC  A           JMP ROUT
FFDB 1FFF  A  DISP:    BOC C15,DISP
FFDC 8300  A           LD 0,(3)
FFDD CDD2  A           ADD 3, ONE;    INCREMENT ADDRESS
FFDE 21D8  A           JMP ROUT
FFDF          ;
FFDF A002  A  SAVE:    ST 0,X'02      ;  SAVE AC0 - AC3 IN LOCATIONS X'02 - X'05.
FFE0 A403  A           ST 1,X'03
FFE1 A804  A           ST 2,X'04
FFE2 AC05  A           ST 3,X'05
FFE3 0080  A           PUSHF
FFE4 4500  A           PULL 1
FFE5 A406  A           ST 1,X'06;     SAVE FLAGS IN LOCATION 6.
FFE6 0200  A           RTS 0
FFE7          ;
FFE7 8406  A  RSTOR:   LD 1,X'06
FFE8 4100  A           PUSH 1
FFE9 0280  A           PULLF
FFEA 8002  A           LD 0,X'02;     RESTORE ACCUMULATORS.
FFEB 8403  A           LD 1,X'03
FFEC 8804  A           LD 2,X'04
FFED 8C05  A           LD 3,X'05
FFEE 0200  A           RTS 0
FFEF                   .PAGE          'INTERRUFT SERVICE ROUTINE FOR HALT AND STACKFULL'
FFEF          ;
FFEF 180B  A  INTR:    BOC C8,STFL
FFF0 29EE  A           JSR SAVE;      SAVE ACCUMULATORS IN LOCATIONS 2,3,4 AND 5.
FFF1 4400  A           PULL 0
FFF2 4000  A           PUSH 0
FFF3 0600  A           ROUT 0;        INTERRUPT ADDRESS IS DISPLAYED.
FFF4 0000  A           HALT
FFF5 0400  A           RIN 0;         READ SWITCHES
FFF6 1102  A           BOC C1,.+3;    IF SWITCHES ARE SET TO ALL ZEROS, 'EXECUTE' CAUSES
FFF7 29EF  A           JSR RSTOR;     A RETURN TO THE CONTROL PANEL ROUTINE.
FFF8 0100  A           RTI ;          IF SWITCHES ARE SET TO ANY NON-ZERO NUMBER, 'EXECUTE
FFF9          ;                       CAUSES A NORMAL RETURN FROM INTERRUPT.
FFF9 4400  A           PULL 0
FFFA 21BB  A           JMP SET;       RETURN TO CONTROL PANEL ROUTINE.
FFFB 4CFF  A  STFL :   LT 0,-1
FFFC 0600  A           ROUT 0         ;  NON RECOVERABLE SITUATION; "INITIALIZE" RETURNS
FFFD 0000  A           HALT           ;  CONTROL TO PANEL.
FFFE 21B2  A           JMP BEGIN
FFFF 0006  A  LAST 6:  .WORD 6
 000                   .END
```

## 2.4 CONTROL PANEL WIRE LIST

The wire list for the connections between the control panel card and the IMP-16C is provided in table 2-1.

Note that the switches from the control panel are brought out directly to terminals. When these lines are connected to the IMP-16C, no other lines may be connected to the same bus. Therefore, if it is desired to hook up other equipment on the input data bus, the switch lines from the control panel must be passed through buffers with tri-state outputs that may be disabled.

Table 2-1. Wire List

| SIGNAL | FROM | TO | SIGNAL | FROM | TO | SIGNAL | FROM | TO |
|---|---|---|---|---|---|---|---|---|
| | Control Panel | IMP-16C | | Control Panel | IMP-16C | | Control Panel | IMP-16C |
| Ground | E1 | 1-4, 141-144 | BDO 05 | E17 | 56 | BDO 13 | E33 | 74 |
| +5v | E2 | 5-8, 137-140 | BDO 06 | E18 | 65 | BDO 15 | E34 | 70 |
| SW 00 | E3 | 79 | BDO 07 | E19 | 63 | BDO 14 | E35 | 69 |
| SW 01 | E4 | 84 | SW 08 | E20 | 90 | WRPB | E36 | 53 |
| BDO 00 | E5 | 60 | SW 09 | E21 | 92 | EXEC | E37 | 123 |
| BDO 01 | E6 | 58 | SW 11 | E22 | 93 | INTRA (1) | E38 | 15 |
| SW 02 | E7 | 81 | SW 10 | E23 | 91 | LDM (2) | E39 | 119 |
| BDO 02 | E8 | 64 | BDO 09 | E24 | 67 | HLT* (3) | E40 | 125 |
| BDO 03 | E9 | 61 | BDO 08 | E25 | 27 | LDA (4) | E41 | 129 |
| SW 03 | E10 | 86 | BDO 10 | E26 | 73 | SYSCLR (5) | E42 | - |
| SW 04 | E11 | 98 | BDO 11 | E27 | 68 | SYSCLR* | E43 | 126 |
| SW 05 | E12 | 103 | SW 12 | E28 | 95 | C 45 (6) | E44 | -99 |
| SW 07 | E13 | 105 | SW 13 | E29 | 96 | DISP (7) | E45 | 107 |
| SW 06 | E14 | 100 | SW 15 | E30 | 106 | -12v | E46 | 31, 32 |
| WRPA | E15 | 66 | SW 14 | E31 | 97 | GROUND | E47 | 1-4, 141-144 |
| BDO 04 | E16 | 59 | BDO 12 | E32 | 75 | -12v SWITCHED | E48 | 11, 12 |

NOTES:
(1) INTERRUPT; WIRED TO CONTROL PANEL HALT
(2) WIRED TO JC13 ON IMP-16C
(3) HALT INDICATOR FLAG
(4) WIRED TO JC12 ON IMP-16C
(5) UNUSED
(6) C45 TIMING SIGNAL
(7) WIRED TO JC15 ON IMP-16C

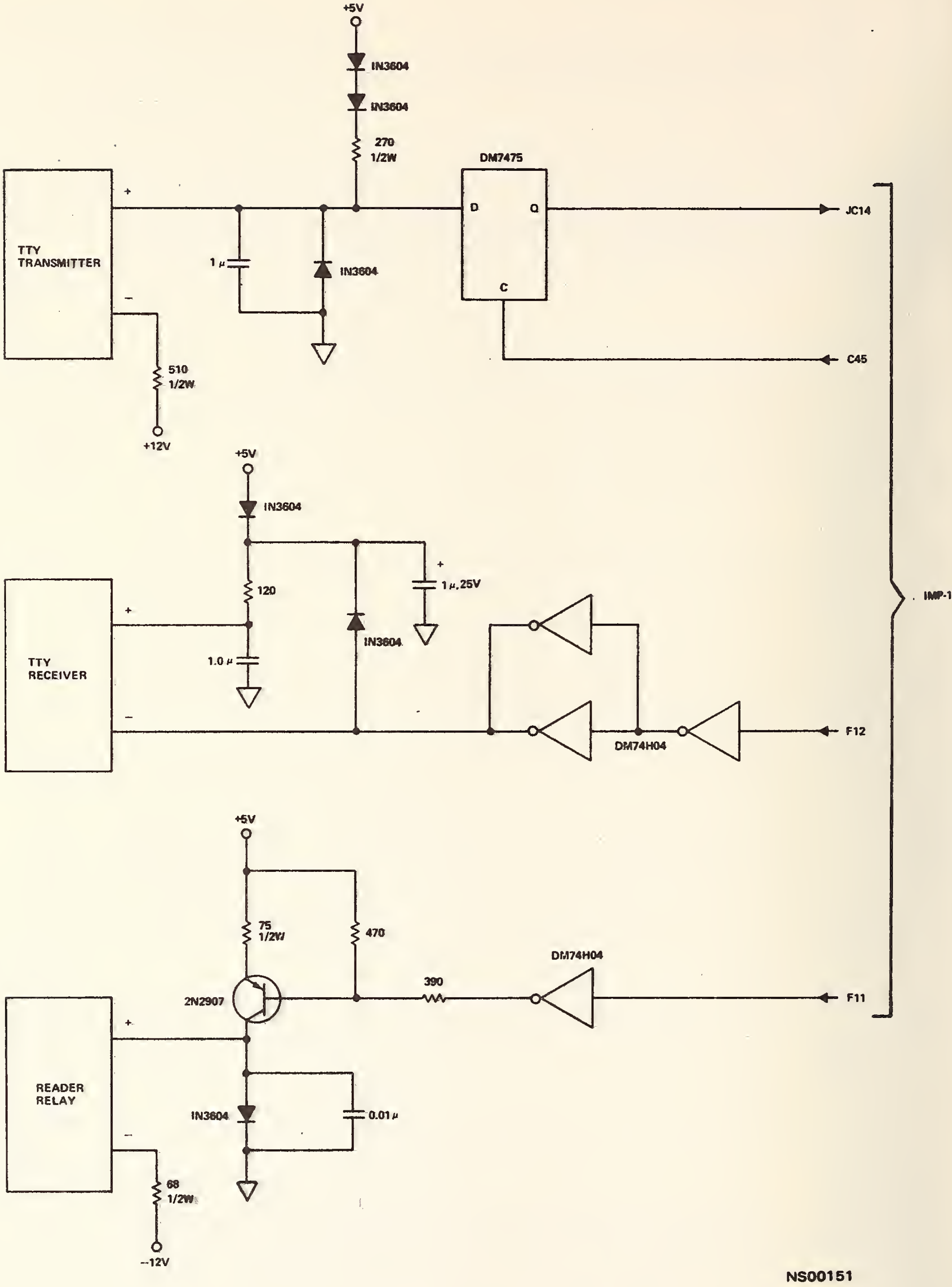


Figure 3-1. Serial Teletype Interface, Flag Controlled

# CHAPTER 3

# SERIAL TELETYPE INTERFACES

## 3.1 GENERAL INFORMATION

Two program-controlled teletype interfaces are presented in this chapter. The first is a simple, economical interface that employs control flags. The second interface requires more hardware since it employs device address decoding, but this interface has the advantage of being completely software-supported and does not tie up user flags. Each of these, along with sample programs, is described in the following paragraphs.

## 3.2 PROGRAM-CONTROLLED INTERFACE USING FLAGS

A very simple teletype interface can be built using two thirds of a DM7404 package and a few discrete elements. This interface permits communication with the IMP-16C via a user jump condition and two user control flags. This approach eliminates the need for device address decoding and,consequently, turns out to be a very economical implementation. Since the address fields of the RIN and ROUT instructions are not used, AC3 is not tied up any more and can be used freely by the programmer. The circuit schematic is given in figure 3-1.

The listings provided under paragraphs 3.2.1 and 3.2.2 describe a character-read routine (RECV) and a character-transmit routine (SEND) that receives/sends bit-serial information between the TTY and the IMP-16C. User jump condition 14 (JC14) is used for data-in and flag 12 (F12) is used for data-out. An additional flag (F11) is used as a reader-enable control signal.

NOTE

1. In figure 3-1, an optional 7475 is used to synchronize the jump condition input into the IMP-16C to assure that the signal is stable between successive T4 times of any microcycle sequence. This is recommended to prevent the rare (but possible) occurrence of the jump condition input making a transition during the leading edge of phase 2. On the IMP-16C/200, C/300 this circuit is already provided.

2. The mnemonic TTY is used frequently to denote "teletype."

### 3.2.1 Teletype Transmit Character Routine

This routine takes one character (right justified in AC0) and sends it to the TTY. Since this is written as a subroutine, accumulators 0, 1, and 2 are saved on the stack before being used in the routine as temporary storage areas.

```
        .PAGE   'TRANSMIT CHARACTER ROUTINE'
        XMIT = 4                    ; TELETYPE TRANSMIT FLAG
SEND:   PUSH    2                   ; SAVE ACCUMULATORS.
        PUSH    1
        SFLG    XMIT                ; SEND START BIT.
        JSR     DELAY               ; DELAY INTO FIRST DATA BIT.
        LI      2,8                 ; SET BIT COUNT.
PUT:    PFLG    XMIT                ; CLEAR TRANSMIT FLAG.
        BOC     3,$XX               ;
        SFLG    XMIT                ; SEND DATA BIT.
$XX:    JSR     DELAY
        SHR     0,1
        AISZ    2,-1                ; TEST TO SEE IF DONE.
        JMP     PUT
        PFLG    XMIT                ; SEND TWO STOP BITS.
        JSR     DELAY
        JSR     DELAY
        PULL    1
        PULL    2
        RTS
```

The DELAY subroutine below is used in the routine above and in the RECV routine to provide the required delay between the teletype bits that are being processed serially.

```
DELAY:  LD      1,V2            ; LOAD TIMING PARAMETER
        AISZ    1,-1
        JMP     .-1
        RTS
V1:     .WORD   01B1
V2:     .WORD   035E
```

3.2.2 Teletype Receive Character Routine

This routine takes one character from the TTY and loads it into AC0 (right justified).

A loader that reads IMP-16 Assembler-generated load modules (RLMs) and that loads memory is described in chapter 5. The programs described in section 3.2 may be used in conjunction with the control panel described in chapter 2.

```
        .PAGE   'TELETYPE GET CHARACTER ROUTINE'
        JC14  = 14                  ; INPUT JUMP CONDITION.
        READR =  3                  ; READER ENABLE FLAG.
        C1    =  1                  ,
        C2    =  2
RECV:   PUSH    1                   ; SAVE ACCUMULATORS.
        PUSH    2
        PFLG    2                   ; DISABLE LINK.
        LI      2,8                 ; SET COUNT FOR 8 BITS.
        SFLG    READR
        BOC     JC14,.+2            ; TEST FOR START BIT.
        JMP     .-1                 ; LOOP UNTIL FOUND.
        LD      1,V1                ; LOAD TIMING PARAMETER.
        JSR     DELAY+1             ; DELAY HALF BIT TIME.
        PFLG    READR
        BOC     JC14,.+2            ; TEST FOR DATA BIT.
        JMP     RECV+2
REP:    JSR     DELAY
        SHR     0,1
        BOC     JC14,.+2
        OR      0,H8000
        AISZ    2,-1                ; DECREMENT COUNT.
        JMP     REP
        JSR     DELAY
        SHR     0,8
        PULL    2
        PULL    1
        RTS

H8000:  .WORD   08000               ; MASK WORD
```

## 3.3 PROGRAM-CONTROLLED INTERFACE USING DEVICE ADDRESSES

A more-conventional (in terms of minicomputer type of usage) TTY interface uses a peripheral device address decoder. This approach obviously requires more hardware, but it has the advantage of being software-supported because IMP-16L TTY routines may be used directly and does not tie up any user flags.

This teletype interface is a full-duplex, bit-serial communication path that allows the processor to send or receive serial bit streams to and from the teletype. The formatting and timing of the bit stream must be controlled by the processor. All communication between the processor and the interface is over the system data bus.

Figure 3-3 shows the hardware required for this type of interface. The driver circuits are the same as described in 3.2. The 8-bit device address and 3-bit order field are decoded by DM7430 gate and a DM74155 3-to-8-line decoder.

The data received by the teletype may be read over the system data bus in bit position 15 by use of the RIN instruction. The data format is shown in figure 3-2. Bits 0 through 14 are undefined and should be masked by the program. The bits comprising the character will be input serially, least significant bit first.

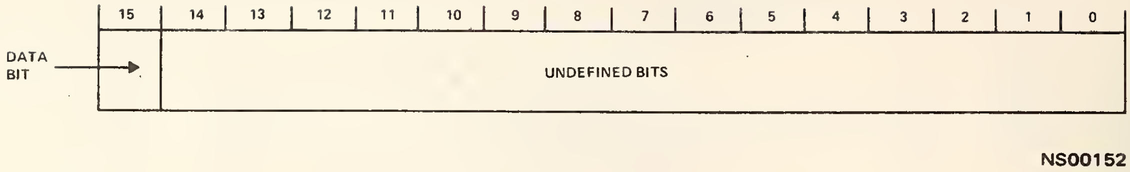


Figure 3-2. Teletype Data Word Format

The teletype transmit circuits may be instructed to transmit a 1 or a 0 by use of the ROUT instruction. Once set to a value, the circuits continue to transmit that value until instructed to transmit another value or cleared with a RESET order. The transmit value is set from bit 15 of the data bus. Bits 0 through 14 are not used and may be any value. The bits comprising the character should be transmitted serially, least significant bit first.

The paper tape reader control may be used for teletypes that have a paper tape reader control circuit. This optional feature allows program control of the tape reader; this is especially desirable for applications where the data are processed as they are read as the reader may be stopped during data processing. Six order codes are acknowledged by the teletype interface. These are listed in table 3-1. The effect of each order is explained below. Those orders marked with an asterisk are not used in the program examples given here.

RIN Code 2 - Bit 15 of the data bus is set equal to the output of the teletype. The processor, in turn, will transfer the data bus to AC0. A binary 1 represents a teletype mark, and a binary 0 represents a teletype space. Bits 0 through 14 are undefined.

RIN Code 4 - The paper tape reader is turned on.

RIN Code 5 - The interrupt request and interrupt enable flags are turned off. The teletype output is set to the idle (marking) state. The paper tape reader enable is turned off.

*RIN Code 6 - The teletype responds to the "Interrupt Select Status 1" order by setting data bit 7 equal to the state of the interrupt request flag.

*ROUT Code 1 - The interrupt enable flag is turned on.

ROUT Code 3 - The. teletype transmit circuit is set to the value of data bit 15.

ROUT Code 4 - The paper tape reader is turned on.

ROUT Code 5 - The interrupt request and interrupt enable flags are turned off. The teletype output is set to idle (marking) state. The paper tape reader enable is turned off.

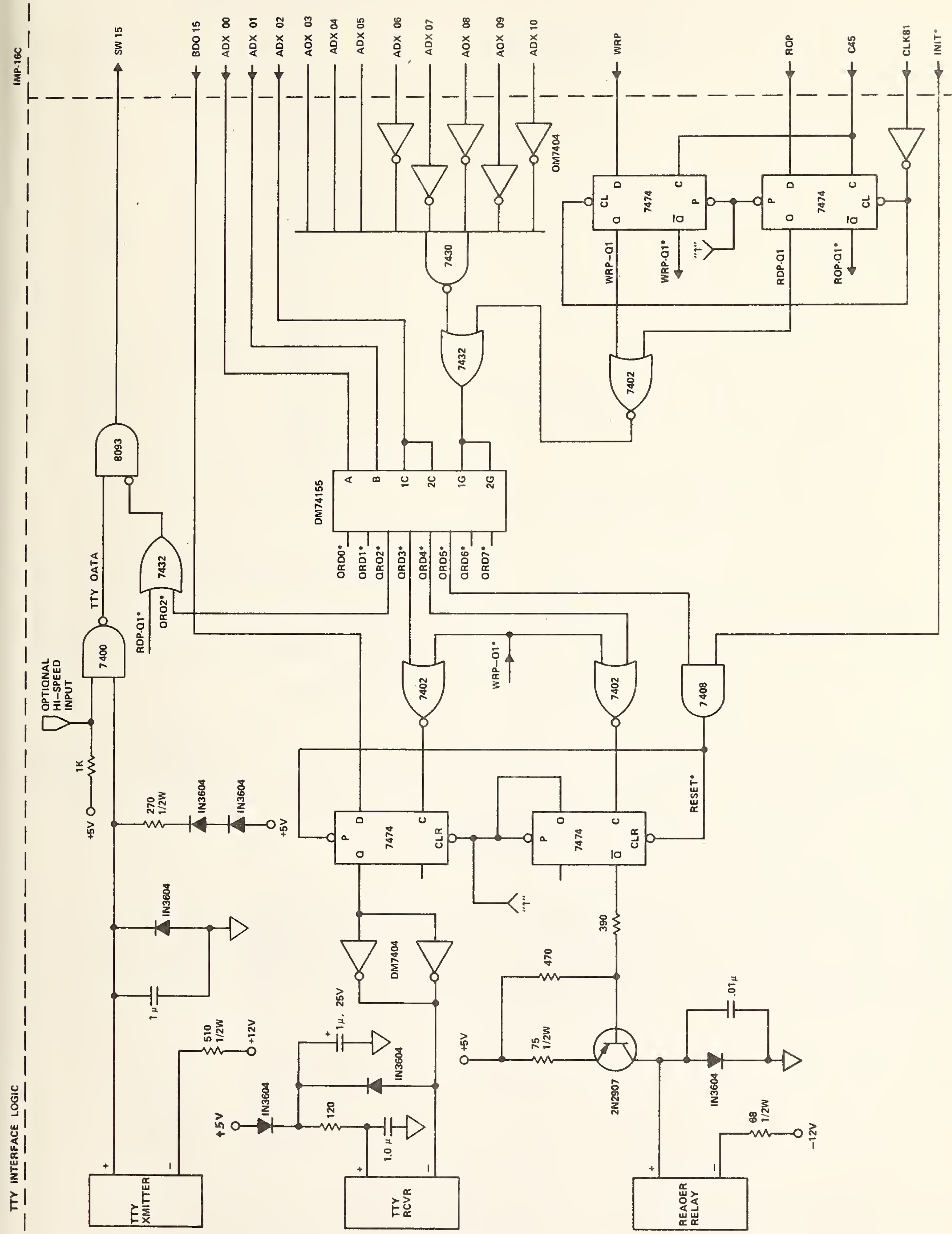


Figure 3-3. Serial Teletype Interface, Device Address-Controlled

Table 3-1. Serial Teletype Order Codes

| Instruction | Order Code | Action |
|---|---|---|
| RIN | 000 | NO ACTION |
| | 001 | NOT ALLOWED |
| | 010 | READ BIT FROM TTY |
| | 011 | NOT ALLOWED |
| | 100 | START TAPE READER |
| | 101 | RESET |
| | 110 | INT STATUS TO BIT 7* |
| | 111 | NO ACTION |
| ROUT | 000 | NO ACTION |
| | 001 | SET INTEN = DATA BIT 15* |
| | 010 | NOT ALLOWED |
| | 011 | WRITE BIT TO TTY |
| | 100 | START TAPE READER |
| | 101 | RESET |
| | 110 | NOT ALLOWED |
| | 111 | NO ACTION |

The program listings that follow show the corresponding receive and transmit routines for this hardware. For these programs, the teletype has been assigned a peripheral address of $0038_{16}$, and the various order codes are given in table 3-1.

### 3.3.1 Teletype Transmit Character Routine

This routine takes one character (right justified in AC0) and sends it to the TTY. Since this is written as a subroutine, the contents of accumulators 1 and 2 are saved in memory before the accumulators are used in the routine as temporary storage areas.

```
        .PAGE   'TELETYPE PUT CHARACTER ROUTINE'
        TTYAD = 7*8              ; TELETYPE ADDRESS
        SEND  = 3
PUTC:   ST      1,7              ; SAVE ACCUMULATORS.
        ST      2,8              ;
        PFLG    2
        LD      1,V1
        JSR     DELAY+1
        LI      2,9
        LI      3,TTYAD
        ROUT    SEND
LP2:    JSR     DELAY
        AISZ    2,-1             ; CHECK TO SEE IF DONE
        JMP     .+2
        JMP     DONE
        ROR     0,1
        ROUT    SEND
        JMP     LP2
DONE:   LI      0,-1             ; SEND STOP BIT.
        ROUT    SEND
        JSR     DELAY
        LD      1,7
        LD      2,8
        RTS                      ; RESTORE ACCUMULATORS.
```

### 3.3.2 Teletype Receive Character Routine

This routine takes 8 bits of serial data from the teletype transmitter interface and packs them into AC0 with the bits right-justified.

```
        .PAGE   'TELETYPE GET CHARACTER ROUTINE'
        TTYAD = 7*8
        RESET = 5
        RDREN = 4
        READ  = 2
GETC:   ST      2,8              ; SAVE AC1 AND AC2 IN
        ST      1,7              ; LOCATIONS 7 AND 8.
        PFLG    2
        LI      3,TTYAD
        ROUT    RESET
        LI      2,8              ; SET BIT COUNT TO 8.
        ROUT    RDREN
        RIN     READ
        BOC     2,.+2            ; TEST FOR START BIT
        JMP     .-2
        LD      1,V1
        JSR     DELAY+1
        RIN     READ
        BOC     2,.+2
        JMP     GETC+3
```

(listing continued)

```
LP1:     JSR     DELAY2          ; DELAY ONE BIT TIME.
         RIN     READ
         AND     0,MASK
         SHR     1,1
         RXOR    0,1
         AISZ    2,-1;           ; TEST TO SEE IF DONE.
         JMP     LP1
         JSR     DELAY           ; DELAY INTO FIRST STOP BIT.
         SHR     1,8
         RCPY    1,0
         LD      1,7             ; RESTORE ACCUMULATORS.
         LD      2,8;
         RTS
MASK:    .WORD   X'8000
```

The GETC and PUTC routines use the same type of DELAY subroutine as the RECV and SEND routines of the previous section. Locations 7 and 8 of main memory are used as temporary storage locations.

```
DELAY:   LD      1,V2
         AISZ    1,-1
         JMP     .-1
         RTS
DELAY2:  LD      0,V2
         AISZ    0,-1
         JMP     .-1
         RTS
V1:      .WORD   433             ; DELAY PARAMETERS
V2:      .WORD   862
```

### 3.3.3 Teletype Get Character and Echo Routine

This routine takes 8 bits of serial data from the teletype transmitter, packs them into AC0 with the bits right-justified, and also echoes the character back to the teletype receiver. The echo operation is done bit-by-bit so that the maximum rate of character processing can be achieved. The program listing follows on page 3-9.

## 3.4 TELETYPE TIMING PARAMETERS

All the teletype programs described in this chapter utilize software delay routines to time the serial transmission of teletype data bits. These routines have timing parameters that provide 1/2-bit and 1-bit delays based on a teletype speed of 110 bits/second. The following example shows how the parameter V1 in the delay routine is derived. A similar calculation yields V2.

Time for AISZ instruction = $4 \times 1.4 + 0.35\ \mu s = 5.95\ \mu s$

Time for JMP instruction = $3 \times 1.4 + 0.35\ \mu s = 4.55\ \mu s$

1/2-bit delay time = 4.545 ms

$$V1 = \frac{4.545 \times 10^{-3}}{(5.95 + 4.55)10^{-6}} = 433$$

# SUPPLEMENT 1

```
        .PAGE   'TELETYPE GET CHARACTER ROUTINE WITH ECHO'
;
TTYAD   =       7*8
READ    =       2
SEND    =       3
RDREN   =       4
RESET   =       5
;
GECHO:  ST      AC2,8              ; SAVE AC1 AND AC2 IN
        ST      AC1,7              ; LOCATIONS 7 AND 8
        PFLG    2
        LI      AC3,TTYAD
        ROUT    RESET              ; RESET TELETYPE
        LI      AC2,8              ; SET BIT COUNT TO 8
        ROUT    RDREN              ; ENABLE READER
        RIN     READ
        BOC     2,.+2              ; TEST FOR START BIT
        JMP     .-2
        LD      AC0,V1
        JSR     DELAY0+1           ; DELAY 1/2 BIT TIME
        RIN     READ               ; TEST IF START BIT IS STILL THERE
        BOC     2,.+2              ; BRANCH IF GOOD START BIT
        JMP     GECHO+3
LP3:    ROUT    SEND               ; ECHO BIT
        JSR     DELAY0             ; DELAY ONE BIT TIME
        RIN     READ
        AND     AC0,MASK           ; MASK UNWANTED BITS
        SHR     AC1,1              ; SHIFT DATA
        RXOR    AC0,AC1            ; ADD NEW BIT TO DATA
        AISZ    AC2,-1             ; TEST TO SEE IF DONE
        JMP     LP3
        ROUT    SEND               ; ECHO LAST BIT
        JSR     DELAY0             ; DELAY INTO FIRST STOP BIT
        LI      AC0,-1
        ROUT    SEND               ; SEND STOP BIT
        SHR     AC1,8              ; SHIFT DATA INTO RIGHT 8 BITS
        RCPY    AC1,AC0            ; PUT CHARACTER IN AC0
        LD      AC1,7              ; RESTORE ACCUMULATORS
        LD      AC2,8
        RTS




MASK:   .WORD   X'8000
V1:     .WORD   01B1
V2:     .WORD   035E
DELAY0: LD      AC0,V2             ; DELAY SUBROUTINE (AC0)
        AISZ    AC0,-1
        JMP     .-1
        RTS
```

TIMING CHART

| MODEL | A (msec) | B (msec) | C (μsec) | D (μsec) | E (msec) |
|---|---|---|---|---|---|
| M200 | 24 | 6.25 | 1314 | 2014 | 8.05 |
| M300 | 24 | 2.60 | 435 | 870 | 102.66 |
| M600 | 24 | 2.60 | 435 | 870 | 3.48 |

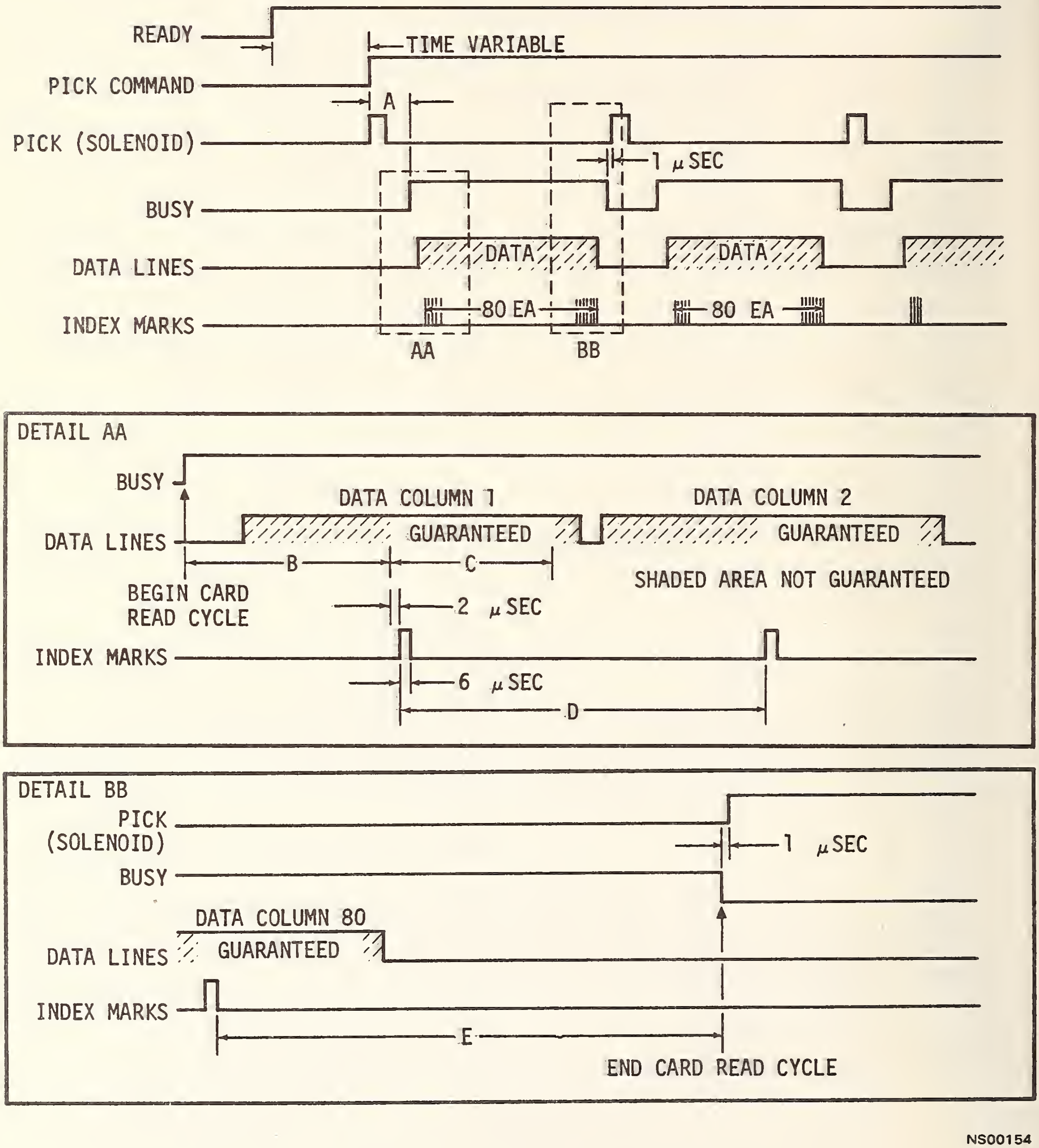


Figure 4-1. Standard Interface Timing for Documation M Card Readers

# CHAPTER 4

# CARD READER INTERFACE

## 4.1 GENERAL INFORMATION

Two simple program-controlled card reader interfaces are described in this chapter. The DOCUMATION 600 (or 300) card reader has been used in the examples; timing for this card reader is shown in figure 4-1. The IMP-16C processor initiates the operation by sending out a "pick" command to fetch a card. After the card has been picked, the card reader sends out 80 index marks that signify the start of each column of data. The IMP-16C program detects the presence of these marks and reads and stores each column of data into an 80-word buffer.

There are two ways to effect this operation: one method that uses only five IC packages makes use of two control flags and no device address decoding. This approach has the advantage of being economical and simple to implement, but it ties up two of the six available user control flags. The second method does not use any flags but requires more hardware to achieve the same purpose. Each of these two methods is described in the following paragraphs.

## 4.2 PROGRAM CONTROLLED INTERFACE USING CONTROL FLAGS

The following program segment shows how to read an 80-column card and store the data into a buffer. The hardware for this is shown in figure 4-3. For this method, no peripheral addresses are required; as a matter of fact, no RIN or ROUT instructions are required for anything other than actual input of data. All other control operations are effected with the SFLG and PFLG instructions.

The circuits shown in figure 4-3 are used with the control panel of chapter 2. When used in this mode, the switches from the control panel are wired into the SW inputs of the DM8123 multiplexers (as shown in figure 4-3) instead of directly to the SW lines of the IMP-16C card. Flag 14 (under program control in the RDCARD routine) selects either the switches or the card-reader for data input to the IMP-16C. The block diagram below shows the arrangement of this interface and a control panel with the IMP-16C.

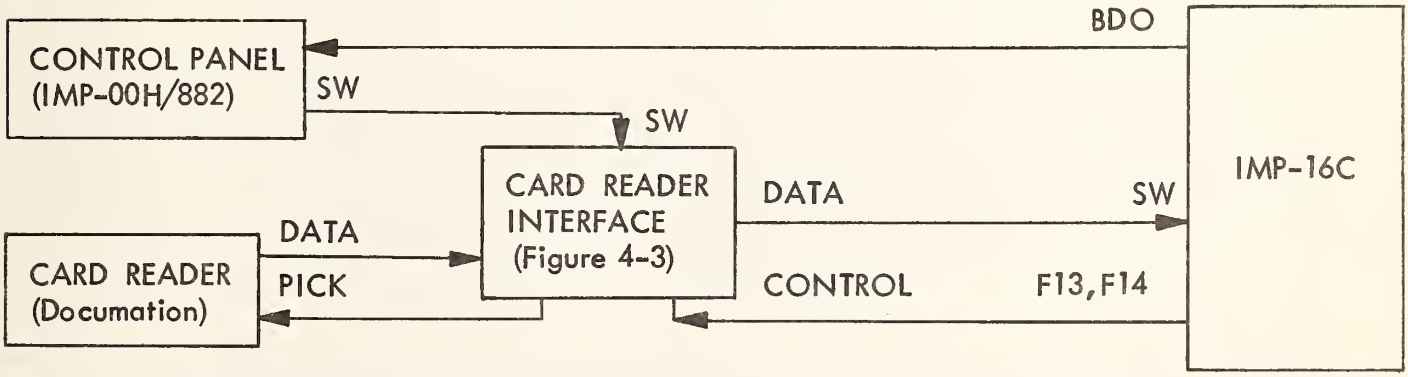


Figure 4-2. IMP-16C and Control Panel Interface, Block Diagram

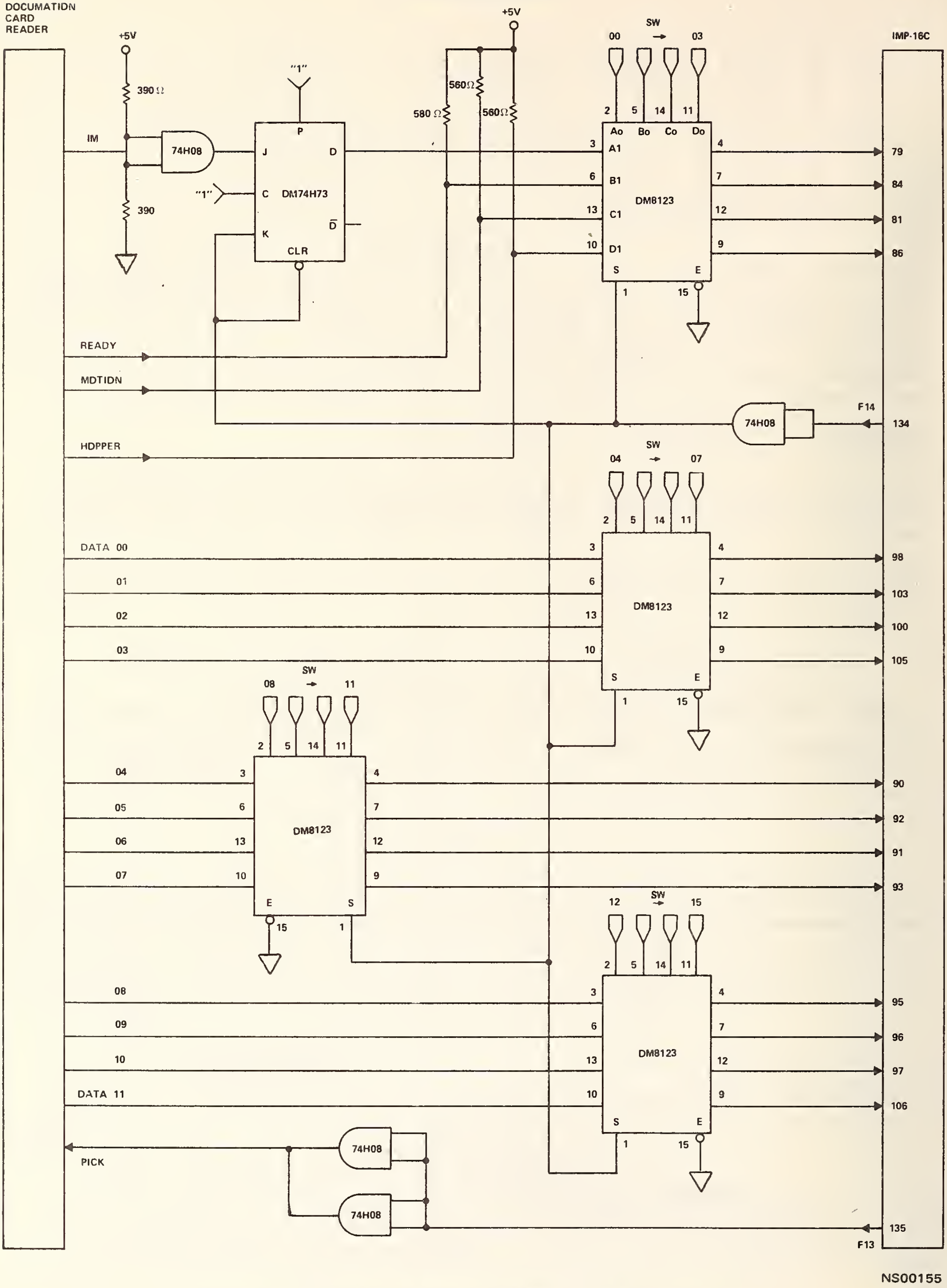


Figure 4-3. Card Reader Interface, Flag-Controlled

```
          BIT0 = 3
          BIT1 = 4
          PICK = 5
          READCARD = 6
          BUFFER = 256 - 80

RDCARD:   PUSH     2
$2:       LD       3,ABUF            ; LOAD ADDRESS OF BUFFER.
          LI       2,80              ; SET COLUMN COUNT.
          SFLG     PICK              ; ENABLE PICK FLAG.
$0:       SFLG     READCARD          ; ENABLE DATA INPUT BUFFERS.
          RIN                        ; GET DATA.
          BOC      BIT0,$1           ; TEST INDEX MARK.
          SKAZ     0,$C              ; TEST HOPPER AND
          JMP      INERR             ; MOTION CHECK.
          BOC      BIT1,$0           ; TEST READY.
INERR:    RCPY     R0,R1
          PFLG     PICK
          LI       R0,1
          HALT
          JMP      $2
$C:       .WORD    0C
$1:       PFLG     PICK              ; START COLUMN DATA PROCESSING,
          PFLG     READCARD          ; DISABLE PICK AND INDEX FF.
          SHR      0,4               ; STRIP STATUS BITS.
          ST       0,(3)             ; SAVE CHARACTER IN BUFFER.
          AISZ     3,1               ; INCREMENT BUFFER ADDRESS.
          AISZ     2,-1              ; DECREMENT COLUMN COUNT.
          JMP      $0                ; JUMP TO READ NEXT COLUMN
          PULL     2
ABUF:     .WORD    BUFFER            ; ADDRESS OF BUFFER HERE.
```

## 4.3 PROGRAM CONTROLLED INTERFACE USING DEVICE ADDRESSES

This approach is a more-conventional approach to I/O interfacing because it makes use of decoded device addresses provided by RIN and ROUT instructions. The number of ICs required increases slightly.

The program listing below describes a read card routine that works with this interface. The hardware is shown in figure 4-4.

Note that this interface does not depend on the use of the simple control panel because a unique device address has been assigned to the card reader. All peripheral devices (including any type of control panel) would communicate with the IMP-16C card via a tri-state input bus connected to the SW lines. In order words, only one device at a time controls the input bus as selected by the appropriate device address. The control panel (switches and lights) would have its own device address.

```
        .PAGE   'READCARD ROUTINE'
        CRADR = 02*8            ; CARD READER ADDRESS
        PICK  = 2               ; PICK COMMAND ORDER CODE.
        RESET = 3               ; RESET PICK FLIP-FLOP.
        READ  = 1               ; READ DATA
RDCARD: PUSH    2
        PUSH    1
        LD      2,ABUF
        LI      3,CRADR         ; LOAD CARD READER ADDRESS.
        LI      1,80
        ROUT    PICK            ; GET CARD
$STRT:  RIN     READ            ; GET DATA
        BOC     BIT1,.+2        ; LOOP UNTIL READY.
        JMP     .-2
        BOC     BIT0,COL        ; TEST FOR INDEX MARK.
        SKAZ    0,$C            ; TEST FOR HOPPER/MOTION CHECK.
        JMP     MOTERR
        JMP     $STRT
COL:    ROUT    RESET
        SHR     0,4             ; STRIP STATUS BITS.
        ST      0,(2)           ; SAVE DATA IN BUFFER
        ADD     2,$ONE          ; INCREMENT BUFFER ADDRESS
        AISZ    1,-1            ; DECREMENT COLUMN COUNT
        JMP     $STRT
        PULL    1
        PULL    2
        RTS
$C:     .WORD   0C
MOTERR: RCPY    0,1
        ROUT    RESET
        LI      0,1
        HALT
ABUF:   .WORD   0               ; ADDRESS OF BUFFER HERE.
$ONE:   .WORD   1
```

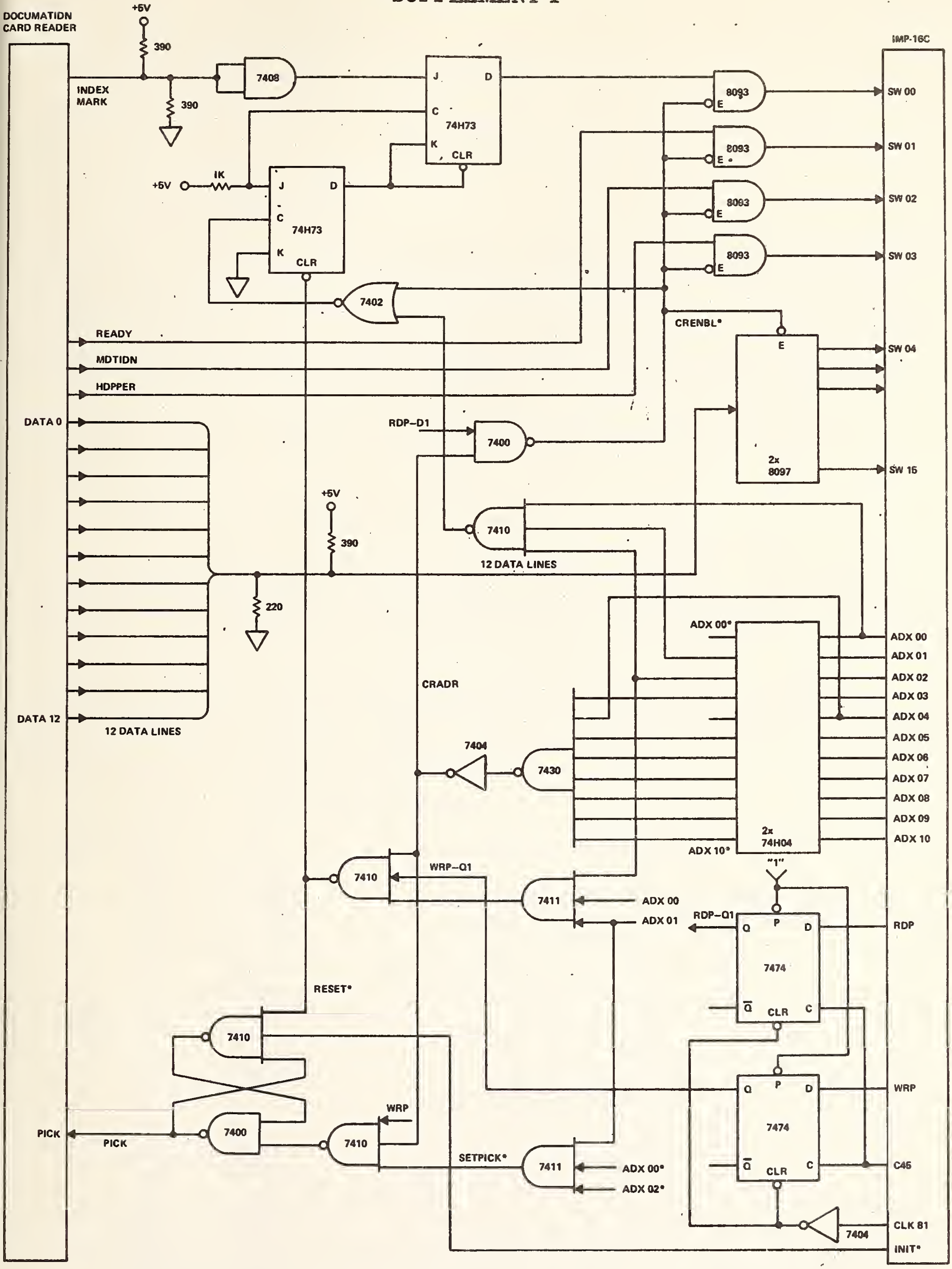


Figure 4-4. Card Reader Interface, Device Address-Controlled

NS00156

# CHAPTER 5

# LOADER ROUTINES

## 5.1 GENERAL INFORMATION

The IMP-16 Assembler creates load modules (tapes or cards) from source programs written in assembly language. These load modules (called RLMs) contain the object code for the program to be executed, relocation information, and other loading information, arranged in four kinds of records: a "title" record, one or more "symbol" records, one or more "data" records, and an "end" record. Specific details of the various record formats are given in the IMP-16 Assembler Manual (4200002).

The loader programs described in the following paragraphs may be used with the routines and programs presented in chapters 3 and 4.

## 5.2 ABSOLUTE LOADER (ABSLDR)

The following loader program takes an absolute paper tape (in RLM format) and loads it into memory from a starting address defined on the tape. After loading the program, the IMP-16C halts and the program entry address is displayed. If EXECUTE is pressed, the processor jumps to the loaded program and begins execution.

This loader program may be used with any of the TTY routines described in chapter 3. That is, either GETC or RECV and PUTC or SEND may be used. The listings shown here call SEND and RECV, but the JSR calls may be changed appropriately to call PUTC and GETC respectively.

```
        .PAGE   'TTYLDR FROM RLM TAPES'
ABSLDR: JSR     RECV
        AISZ    0,-2            ; LOOK FOR STX CHARACTER.
        JMP     ABSLDR
        JSR     RDPACK2         ; START PROCESSING RECORD CONTROL INFO
        BOC     C2,TORS         ; IF BIT15=1 GO TO TITLE, SYMBL PRCSSING
        SHL     0,1
        BOC     C2,.+2          ; IF BIT14=0 GO TO DATA RECORD.
        JMP     ENDREC
DATREC: SHR     0,1
        RCPY    0,3
        JSR     RDPACK2
        RCPY    0,1
        PUSH    0               ; SAVE CHECKSUM WORD.
        CAI     1,1
        JSR     RDCHK
        JSR     RDCHK
        RCPY    0,2
        JSR     RDCHK
        JSR     RDCHK
        RADD    2,3
        CAI     3,1+4
```

(listing continued)

```
LOAD:   JSR     RDCHK
        ST      0,(2)
        ADD     2,$ONE          ; INCREMENT ADDRESS.
        RCPY    2,0
        RADD    3,0
        BOC     C2,CKTEST
        JMP     LOAD
RDCHK:  JSR     RDPACK2
        RADD    0,1
        RTS
CKTEST: PULL    0               ; CHECKSUM TEST
        BOC     C1,ABSLDR
        RCPY    1,0
        BOC     C1,ABSLDR
        LI      0,X'33
        HALT                    ; CHECKSUM ERROR.
TORS:   AND     0,H3FFF
        RCPY    0,1
        JSR     RDPACK2
        AISZ    1,-1
        JMP     .-2
        JSR     RDPACK2
        JMP     ABSLDR
ENDREC: JSR     RDPACK2
        JSR     RDPACK2
        JSR     RDPACK2
        HALT
RDPACK2:JSR     RECV            ; READS AND PACKS TWO CHAR IN AC0.
        PUSH    1
        SHL     0,8
        PUSH    0
        JSR     RECV
        PULL    1
        RXOR    1,0
        PULL    1
        RTS
```

## 5.3 PAPER TAPE BOOTSTRAP LOADER (PTBOOT)

For programs of short length, it is sometimes desirable to load directly into memory without going through an assembly. For such cases, the following program serves the purpose of loading hexadecimal information directly from paper tape. The loader format is laid out such that the first 4 hexadecimal characters (that is, 4 rows of 8-bit ASCII coded characters) from the paper tape are interpreted as the starting address. All subsequent blocks of 4 characters each are interpreted as data and packed into 16-bit words. The last recognizable character on the tape should be "!". All data characters are arranged so that the most significant bits appear first. The PTBOOT program reads the tape and returns control to the panel routine. This assumes that this program is used with the control panel routine described in chapter 2. PTBOOT calls the teletype RECV routine described in 3.2.2.

PTBOOT PROGRAM

```
LOOP1:   LI 1,4
         JSR PTBOOT
         RCPY 2,3;                  READ FIRST 4 WORDS; THIS IS THE
;                                   STARTING ADDRESS.
LOOP2:   LI 1,4;
         JSR PTBOOT
         ST 2,(3);                  READ AND STORE 4 HEX WORDS.
         ADD 3,ONE;                 INCREMENT MEMORY ADDRESS.
         JMP LOOP2
         LI 0,X'77;                 ERROR CODE X'77
         HALT;                      ATTEMPTED TO LOAD LOC 0.
PTBOOT:  JSR RECV;                  GET ONE WORD.
PACK:    AND 0,MSKPAR;              MASK OUT PARITY BIT.
         SKNE 0,EXCLAM;
         JMP $OUT
         SKNE 0,CRETRN;             IGNORE  CARRIAGE RETURN.
         JMP PTBOOT
         SKNE 0,LINEFD;             IGNORE LINEFEED.
         JMP PTBOOT
         SKAZ 0,NUMBER;
         JMP NUM
         SKAZ 0,ALPHA;
         JMP ALFA
         LI 0,3;                    ERROR CODE: INVALID CHARACTER.
         JMP 0;
ALFA:    ADD 0,NINE;
NUM:     AND 0,MSK1
         SHL 2,4
         RXOR 0,2
         AISZ 1,-1
         JMP PTBOOT
         RTS
$OUT:    PULL 1
         PULL 1
         HALT
NINE:    .WORD 9
MSKPAR:  .WORD 07F
MSK1:    .WORD 0F
NUMBER:  .WORD 030
ALPHA:   .WORD 040
CRETRN:  .WORD 0D
LINEFD:  .WORD 0A
EXCLAM:  .WORD 021
RECV1:   .WORD RECV
```

## 5.4 CARD READER LOADER (CRLM)

This loader program takes an absolute RLM in card format and loads it into memory. It uses 80 words of memory as a temporary buffer. For the example program given here, locations 176 to 255 in base page are used for the buffer. The object deck to be loaded must be followed by a !G card. This loader routine calls a RDCARD subroutine, which may be any one of the two described in chapter 4.

```
;           THIS IS AN ABSOLUTE LOADER FOR THE IMP-16C/P SYSTEM
;
;           ( BASE PAGE VERSION ).
;
;           THIS PROGRAM READS ONE RLM FROM THE CARD READER AND
;           LOADS IT INTO MEMORY.  THE RLM MUST HAVE BEEN PUNCHED
;           INTO CARD COLUMNS 1-72, AND CAN CONTAIN PUNCH CODES
;           ONLY FOR THE CHARACTERS 0,1,...,9,A,B,...,F, OR BLANK.
;           BLANKS WILL BE TREATED AS 0.
;
;           THE RLM MUST BE IN STANDARD RLM FORMAT. THE TITLE CARD
;           AND SYMBOL CARDS ARE IGNORED. DATA FROM DATA CARDS IS
;           MOVED TO THE SPECIFIED LOAD LOCATIONS WITHOUT ANY
;           RELOCATION PERFORMED.  THE END CARD MUST CONTAIN AN
;           ENTRY POINT ADDRESS (SEE ERROR CODE 5, BELOW) AND IS
;           LAST CARD READ BY ABSCR.
;
;           A CHECKSUM TEST IS PERFORMED ON ALL INPUT CARDS.  (SEE
;           ERROR CODE 3, BELOW.)
;
;           BUFFER AREA IN BASE PAGE, LOCS. 256-80.
;


;       ERROR   MEANING             ACTION
;       -----   ----------------    -------------------------------------
;         1     I/O ERROR           REPLACE CARD IN READER AND PUSH START
;         2     INV. CHARACTER      CORRECT CARD, REPLACE IN READER, AND
;                                      PUSH START.  (ONLY CODES 0,...,F
;                                      AND BLANK ARE ALLOWED.)
;         3     CHECKSUM ERROR      CORRECT CARD, REPLACE IN READER, AND
;                                      PUSH START.
;         5     INV. ENTRY POINT    SET CORRECT ENTRY POINT INTO REG. 1
;                                      AND PUSH START.
;
;       ALL ERROR CODES ARE LOADED INTO REG. 0 BEFORE HALTING.


        .PAGE   'ABSCR ROUTINE - IMP-16C'
        R0 = 0
        R1 = 1
        R2 = 2
        R3 = 3
        BIT0 = 3
        BIT1 = 4
        NZERO= 5
```

```
INLOOP: JSR  RDCARD;          READ ONE CARD.
        JSR  CNVRT;           PACK DATA.
        LD 3,ABUF;
        LD 0,(3)
        AND 0,MASK3
        RCPY 0,2
        LI 1,0;               SUM=0
        SKNE 1,1(3);          IF CHECKSUM = 0, DATA VALID.
        JMP VALID
CKSUML: ADD 1,2(3)
        AISZ 3,1;             INCREMENT ADDRESS
        AISZ 2,-1;
        JMP CKSUML
        LD 3,ABUF
        SKNE 1,1(3);          COMPARE SUM&CKSUM.
        JMP VALID
ERR3:   LI 0,3;               ERROR CODE 3; CHECKSUM ERROR
        HALT
        JMP INLOOP
VALID:  LD 2,(3)
        SHR 2,14;             ISOLATE CODE FOR RECORD TYPE.
        ADD 2,JTBL            AC2 <- CODE + JUMP TABLE ADDRESS.
        JMP @(2);             JUMP TO PROCESS RECORD TYPE.




TITLE:  JMP INLOOP;           IGNORE TITLE CARD.




SYMBOL: JMP INLOOP;           IGNORE SYMBOL CARD.




END:    LD 1,3(3);             SAVE PROGRAM ENTRY POINT
        ST 1,8;               LOCATION 8 HAS ENTRY POINT
        JMP INLOOP
```

(listing continued)

```
EBUF:    .WORD   BUFFER+72;
BUFFER   = 256 - 80
ATBL:    .WORD   TBL;
ETBL:    .WORD   TBL+15;
JTBL:    .WORD   JUMPTBL;
JUMPTBL:.WORD    TITLE,SYMBOL,DATA,END;
DATA:    LD      R2,3(R3);          A <-- INITIAL LOAD ADDRESS
         AISZ    R0,-4;             L <-- L - 4 (SKIP OVER RELOC. INFO)
         AISZ    R3,6;              P <-- P + 6 (SKIP OVER RELOC. INFO)
DLOOP:   LD      R1,0(R3);
         ST      R1,0(R2);          WORD(A) <-- WORD(P)
         AISZ    R3,1;              P <-- P + 1
         AISZ    R2,1;              A <-- A + 1
         AISZ    R0,-1;             L <-- L - 1, SKIP IF DONE
         JMP     DLOOP;             LOOP FOR NEXT WORD
         JMP     INLOOP;             LOOP FOR NEXT CARD

         .PAGE   'HOLLERITH TO BINARY*CONVERSION'
         ENTRY = 8;                 ENTRY POINT IN LOCATION 8.
CNVRT: LD 3,ABUF;                   FETCH BUFFER ADDRESS.
         RCPY 3,2
         LD 1,(3)
         SKNE 1,EXCLAM;             IF "!G" CARD, GO TO ENTRY POINT
         JMP .+2
         JMP LOOP1
         LD 1,1(3)
         SKNE 1,G
         JMP .+2
         JMP LOOP1
         PULL 0;                    EXIT FROM SUBROUTINE; POP STACK
         LD 2,ENTRY
         SKG 2,ZERO;                IF ENTRY POINT >0, SKIP
         JMP ERROR5;                ELSE ERROR CODE 5.
$EX:      LI 3,1;                   IDENTIFY LOAD DEVICE AS CARD READER
         JMP (2);                   JUMP TO LOADED PROGRAM
ERROR5: LI 0,5;                     ERROR 5: INVALID ENTRY POINT.
ZERO:    HALT
         RCPY 1,2;                  HALT AND THEN
         JMP $EX;                   JUMP TO LOADED PROGRAM.


LOOP1:   LI 0,-4
         LD 1,(3)
         ST 3,7;                    SAVE CURRENT BUFFER LOCATION
         LD 3,ATBL
LOOP2:   SKNE 1,(3)
         JMP STEP2
         ADD 3,TBL+9;               INCREMENT ADDRESS
         SKG 3,ETBL;                SKIP IF END OF TABLE
         JMP LOOP2
         SKNE 1,ZERO;               IF NOT BLANK THEN SKIP.
         JMP BLANK
ERR2:    LI 0,2;                    INVALID CHARACTER.
         HALT;                      HALT AND THEN
         PULL 0;                    REREAD CARD.
         JMP INLOOP
```

```
BLANK:  LD 3,ATBL
STEP2:  SUB 3,ATBL
        LD 1,9
        SHL 1,4
        RXOR 3,1;                    MERGE 4 BITS AT A TIME.
        ST 1,9;                      SAVE PARTIALLY PACKED WORD.
        LD 3,7;                      GET BUFFER LOCATION
        ADD 3,TBL+9;                 INCREMENT BUFFER LOCATION
        AISZ 0,1;                    CHECK TO SEE IF 4 WORDS COLLECTED.
        JMP LOOP1+1
        ST 1,(2);                    STORE PACKED WORD
        ADD 2,TBL+9
        SKNE 3,EBUF
        RTS
        JMP LOOP1



MASK1:  .WORD   X'C0                 ; TRANSMISSION ERROR OR DATA OVERRUN.
MASK3:  .WORD   X'3FFF               ; ALL BITS BUT RECORD TYPE.
THREE:  .WORD   3
EXCLAM: .WORD   X'482
G:      .WORD   X'804
TBL:    .WORD   0200,0100,0080,0040,0020,0010
MASK2:  .WORD   0008                 ; HOLLERITH 6 ALSO ERROR CODE FOR CR BUSY.
        .WORD   0004,0002,0001
        .WORD   0900,0880,0840,0820,0810,0808
        .END    INLOOP
```

# CHAPTER 6

# APPLICATION PROGRAMS

## 6.1 GENERAL INFORMATION

This chapter presents two application programs that can be used to obtain memory dumps onto paper tapes as described in the following paragraphs.

## 6.2 PROM TAPE GENERATOR

This program takes the contents of a specified range of memory locations and dumps them on to paper tape in binary 8-channel format suitable for programming ROMs on the DATA I/O PROM programmer. The first tape generated contains the left byte of the 16-bit word and the second tape contains the right byte. This routine calls the SEND routine of the teletype utilities package. Before executing the program, AC2 should contain the starting address of the range and AC3 the final address. The DATA I/O machine has negative logic, so the data bits are complemented before being punched.

```
; PROM TAPE PROGRAM
; -----------------
BLNK:   LI      1,20
        LI      0,-1
        JSR@    SEND1           ; ROUTINE TO PROVIDE
        AISZ    1,-1            ; LEADING BLANKS
        JMP     .-2
        RTS
        JSR     BLNK
        PUSH    2               ; SAVE STARTING ADDRESS
        LD      1,PROM          ; DUMMY WORD FOR COMPARISON
PROM:   LD      0,(2)
        CAI     0,0             ; INVERT BITS FOR DATA I/O
                                ; MACHINE
        SKNE    1,PROM          ; SKIP FOR RIGHT SIDE
        SHR     0,8             ; RIGHT JUSTIFY MSBYTE
        JSR@    SEND1
        RCPY    3,0             ; CHECK TO SEE IF DONE
        RXOR    2,0
        BOC     1,.+3           ; IF DONE GO TO HALT
        AISZ    2,1             ; INCREMENT ADDRESS
        JMP     PROM
        HALT                    ; AWAIT "EXECUTE" FOR
                                ; RIGHT SIDE
        JSR     BLNK
        PULL    2               ; RESTORE ADDRESS
        JMP     PROM            ; GENERATE RIGHT SIDE TAPE
SEND1:  .WORD   0FF53           ; ADDRESS OF SEND ROUTINE IN CUTIL
```

## 6.3 PAPER TAPE PUNCH PROGRAM

This program generates an absolute paper tape from the contents of a specified range in memory. The first four characters on the tape are the ASCII equivalent of the four hexadecimal numbers that specify the starting address. The last character is an exclamation mark; this serves as a termination indicator. The listing for this program is given in appendix A.

# CHAPTER 7

# INTERRUPT HANDLING

## 7.1 GENERAL INFORMATION

In the IMP-16C, there are two processor interrupt request lines; one of these is reserved for stack overflow interrupts. All external peripheral devices are wired to the main interrupt request line (INTRA). If any device generates an interrupt request, the line goes high and interrupts the processor if the master interrupt enable (INTEN) is set for the processor.

## 7.2 INTERRUPT RESPONSE

Response to a processor interrupt occurs at the end of the instruction executing at the time the interrupt occurs. The interrupt causes the processor to save the current state of the program counter (PC) on the top of the stack and to set the new contents of the PC equal to 1. The interrupt enable flag (INTEN) is then turned off, and the processor executes the instruction in memory location 1, which is the start of the interrupt service routine. The interrupt service routine may determine the presence of a stack overflow interrupt by use of the BOC instruction. The instruction in location 1 may also be a jump to an interrupt routine located elsewhere.

Available at the card-edge connector of the IMP-16C are two status flags (Flag 0 and Flag 12); these status flags may be used in multi-level interrupt schemes. These flags would then serve as interrupt enable signals for each level of a two-level system. Figure 7-1 shows the external circuits required for this function.

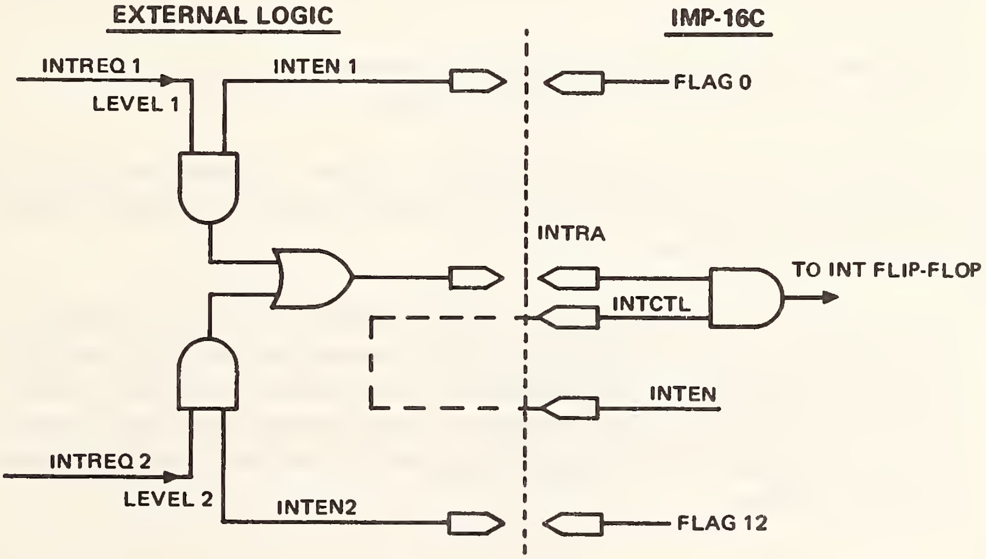


Figure 7-1. Interrupt Response External Circuits

## 7.3 INTERRUPT GENERATION AND PROCESSING

The following discussion pertains to a two-level interrupt system, but all the operations apply to single-level interrupts as well.

### 7.3.1 General Interrupt

Interrupt requests on levels 1 and 2 require that the interrupt service routine determine the address of the interrupting device. This can be done conveniently using the "Interrupt Select Status" order as described below.

A peripheral controller requiring interrupt servicing by the processor sets its interrupt request flag, thus causing a true signal on the interrupt request line if the peripheral's interrupt enable flag is set true. The interrupt request line is common to all peripheral controllers on a given level and only requires that one peripheral device be requesting service in order to set the line to the true state. The main program, when ready, sets the processor INTEN flip-flop, thus indicating that the processor may perform an interrupt service. If both the processor INTEN and the interrupt enable for the given level are enabled at the same time, the processor transfers program control to the interrupt service routine.

When the processor responds to an interrupt request, it goes through the following procedure in order to transfer program control to the interrupt service routine:

1. Transfers the contents of the Program Counter (PC) to the top of the stack.

2. Places the address of memory location 1 into the PC.

3. Disables (clears) the processor INTEN flag to prevent further interrupt.

4. Fetches the next instruction from memory location 1, thus initiating the interrupt service routine.

When an interrupt occurs, the interrupt service routine directs the processor to determine the devices requiring service and to select one peripheral device. This may be accomplished as follows:

1. An "interrupt select status" order is sent out to all peripheral controllers. The order field of the command word is the only field recognized by the peripheral controllers, and the address field is ignored. Upon receipt of the "interrupt select status" order, the INT REQ flags at the peripheral devices are cleared in most cases.

2. Each peripheral device is assigned one of the 16 system bus lines to report its interrupt status. Each peripheral device responds simultaneously with other peripheral devices, indicating whether or not it requires interrupt servicing. A binary 1 indicates a service request. Typical interrupt assignments are shown in table 7-1.

3. The interrupt service routine resolves interrupt priority and selects the peripheral device for interrupt servicing.

NOTE

> Although there are only 16 system bus lines that are used for reporting interrupt status, by use of two "interrupt select status" orders, 1 and 2, the status of 32 peripheral devices may be determined, one group of 16 peripheral devices responding to "interrupt select status 1" and another group to "interrupt select status 2." This concept can be extended to more than two levels of interrupt select status orders if necessary.

After a peripheral device obtains interrupt access, the applicable RIN or ROUT command then effects the transfer of data between the processor and the peripheral controller. Upon completion of the interrupt operations, program control is transferred back to the main program by use of the RTI instruction. The RTI instruction causes the program counter to be loaded by adding the top word of the stack to the CTL field of the instruction. The INTEN flip-flop is then enabled.

Table 7-1. Typical Interrupt Select Status 1 Bit Assignments

| Bit | Assigned Peripheral |
|---|---|
| 0 | Unassigned |
| 1 | Parallel Teletype |
| 2 | Card Reader |
| 3 | Disc |
| 4 | Communications |
| 5 | Interval Timer |
| 6 | Unassigned |
| 7 | Serial Teletype |
| 8 | Modem |
| 9 | Unassigned |
| 10 | Unassigned |

(table continued)

Table 7-1. Typical Interrupt Select Status 1 Bit Assignment (Cont.)

| Bit | Assigned Peripheral |
|---|---|
| 11 | Unassigned |
| 12 | Unassigned |
| 13 | Unassigned |
| 14 | Unassigned |
| 15 | Unassigned |

### 7.3.2 Stack Overflow Interrupt

If the processor stack becomes full, a stack overflow interrupt is set. If the INTEN flag is set, then the processor is interrupted. The interrupt service routine may then test jump condition 8 to determine that the interrupt was caused by a stack overflow.

In most applications, the software may be written in a manner that will guarantee that a stack overflow will not occur. However, even in such cases, it is possible in the system programming development that a stack overflow may accidentally occur. Because the interrupt process itself utilizes the stack, some words of the stack may be lost in the overflow interrupting sequence.

The systems programmer must take some precautions to guarantee that a stack overflow error does not go undetected and that data on the stack are not lost. The programmer must not push words of all zeros onto the stack as data; two protection words of all ones should be kept at the bottom of the stack. These are relatively minor restrictions since, in most applications, use of the stack is reserved for subroutine return addresses, interrupt return addresses, and for saving the RALU flags while servicing interrupts.

## 7.4 SAMPLE PROGRAM FOR INTERRUPT PROCESSING

Consider the case of four peripheral devices connected to the IMP-16C in a priority scheme such that when an interrupt occurs the interrupting device sends over the bus a data word whose value identifies the device. The following program segment is a service routine that determines the identity of the device and sends control to a device routine. After the device has been serviced, the interrupt routine returns control to the main program.

The four devices for this example are a keyboard, an assemblage of digital integrators, a display unit, and a printer unit — arranged such that the keyboard assumes the highest priority. The keyboard indicates its status by sending a high level on data bit 0 of the bus; the digital integrators use bit 1, the display uses bit 2, and the printer uses bit 3. The values returned are, therefore, 1, 2, 4, and 8. The order code for reading status has been chosen arbitrarily as 2.

## SUPPLEMENT 1

In this example, the IMP-16C is the host processor that monitors the operation of the peripheral system components. The digital integrators operate independently, requiring processor intervention only to exchange data, initializing information, and setting up other parameters. The display unit provides a continuous visual display of the operation of the integrators, and the printer provides hard copy when necessary.

```
         INTRAD = X'40
         STATUS = 2
         IX2    = 2
         ACO    = 0


IDTABL:  .WORD.                   ; ADDRESS OF SELF.
         .WORD   KEYBD            ; ADDRESS OF KEYBOARD ROUTINE.
         .WORD   DIGINT           ; ADDRESS OF INTEGRATORS ROUTINE.
         .WORD   BADINT           ; ILLEGAL INTERRUPT ADDRESS.
         .WORD   DISPLA           ; ADDRESS OF DISPLAY ROUTINE.
         .WORD   BADINT           ; ILLEGAL INTERRUPT ADDRESS.
         .WORD   BADINT           ; ILLEGAL INTERRUPT ADDRESS.
         .WORD   BADINT           ; ILLEGAL INTERRUPT ADDRESS.
         .WORD   PRINTR           ; ADDRESS OF PRINTER ROUTINE.
;
;        INTERRUPT SERVICE ROUTINE BEGINS HERE
;
INTSERV: PUSHF                    ; SAVE STATUS FLAGS.
         JSR     SAVE1            ; SAVE ACCUMULATORS IN RAM.
         LI      3,INTRAD
         RIN     STATUS           ; DETERMINE INTERRUPTING DEVICE
;                                     BY READING THE INPUT BUS
;                                     FOR DEVICE STATUS BITS.
         RCPY    ACO,IX2          ; ESTABLISH INDEX VALUE.
         ADD     IX2,IDTABL       ; MAKE A LOCAL ADDRESS.
         JSR@    (IX2)            ; JUMP TO DEVICE SERVICE ROUTINE.
         JSR     RSTOR1           ; RETURN HERE AFTER SERVICING
;                                     DEVICE AND RESTORE ACCUMULATORS.
         PULLF                    ; RESTORE STATUS FLAGS.
         RTI     0                ; RETURN TO INTERRUPTED PROGRAM.
RSTOR1:  LD      0,SVR0
         LD      1,SVR1
         LD      2,SVR2
         LD      3,SVR3
         RTS     0
SAVE1:   ST      0,SVR0
         ST      1,SVR1
         ST      2,SVR2
         ST      3,SVR3
         RTS     0
SVR0:    .WORD   0                ; STORAGE LOCATIONS FOR
SVR1:    .WORD   0                ; ACCUMULATORS.
SVR2:    .WORD   0                ;
SVR3:    .WORD   0
```

(listing continued)

```
KEYBD:   .                              ; SERVICE ROUTINES FOR
         .                              ; THE VARIOUS PERIPHERALS.
         .
DIGINT:  .

DISPLA:  .

PRINTR:  .

BADINT:  .
```

# APPENDIX A

## ASSEMBLY LISTING

An actual assembly listing is presented on the following pages to illustrate some of the topics described in this manual. This program listing consolidates some of the routines described earlier and is a complete firmware package (available in PROM) that can be installed and run on the IMP-16C with the control panel described in chapter 2. The program is assembled at location $FF00_{16}$ and occupies the top page of memory. A few locations in base page read/write memory are used for temporary storage. The following subroutines and procedures are included in the listirg.

1. Control Panel Service Routine
2. Panel HALT Interrupt Routine
3. Teletype Receive Character Subroutine
4. Teletype Transmit Character Subroutine
5. Paper Tape Absolute RLM Loader
6. Paper Tape Loader (BOOTSTRAP)
7. Tape Punch Program

The control panel described in chapter 2 is required for this program package. Additional interface hardware required are the circuits depicted in figure 3-1.

This package is available in ROM under the part name CUTIL. The two teletype routines in this package (SEND and RECV) are written as subroutines so they may be called from user programs with the JSR@ instruction. Entry points and operating instructions are given in the program listings.

```
REVISION-C 11/20/72                                          73227   09143181
CUTIL    CONTROL PANEL AND TTY UTILITIES                     PAGE NUMBER    1


 0000                      .TITLE CUTIL,  'CONTROL PANEL AND TTY UTILITIES'
 0000                      .ASECT
 0000 0001 A               C1 = 1          ;  ACO = 0
 0000 0002 A               C2 = 2          ;  BIT 15 OF ACO IS 0
 0000 0003 A               C3 = 3          ;  BIT 0 OF ACO IS 1
 0000 0004 A               C4 = 4          ;  BIT 1 OF ACO = 1
 0000 0005 A               C5 = 5          ;  ACO ¬= 0
 0000 0007 A               C7 = 7          ;  START CONDITION
 0000 0008 A               C8 = 8          ;  STACK FULL CONDITION
 0000 000B A               C11 = 11        ;  ACO LTEQ 0
 0000 000C A               C12 = 12        ;  JUMP COND.12 (WIRED TO "LOAD ADDRESS")
 0000 000D A               C13 = 13        ;  JUMP COND. 13 (WIRED TO " LOAD DATA")
 0000 000E A               C14 = 14        ;  JUMP COND. 14
 0000 000F A               C15 = 15        ;  JUMP COND. 15 (WIRED TO "DISPLAY")
 0000 FF00 A               .=X'FF00
 FF00           ;
 FF00           ;              THIS UTILITIES PACKAGE CONTAINS A SIMPLE
 FF00           ;              CONTROL PANEL ROUTINE THAT OPERATES IN
 FF00           ;              CONJUNCTION WITH THE CONTROL PANEL KIT
 FF00           ;              (CTLPLKIT),AND SOME TELETYPE ROUTINES
 FF00           ;              TO LOAD PROGRAMS AND PUNCH TAPES.
 FF00           ;


ABSTTY FOR THE IMP-16C


 FF00                      .PAGE    'ABSTTY FOR THE IMP-16C'
 FF00           ;
 FF00           ;                   LOADS ABSOLUTE PAPER TAPE IN RLM FORMAT.
 FF00           ;                   EACH RECORD MUST BE PRECEDED BY STX.
 FF00           ;
 FF00           ;                   ENTRY POINT FOR THIS PROGRAM IS:  FF00
 FF00           ;
 FF00 293C A ABSTTY: JSR RECV
 FF01 48FE A         AISZ 0,-2;                 LOOK FOR START OF TEXT.
 FF02 21FD A         JMP ABSTTY
 FF03 292B A TTY1:   JSR RDPCK;                 PROCESS RECORD CONTROL INFORMATION.
 FF04 121F A         BOC 2,TORS;                BRANCH IF TITLE OR SYMBOL RECORD.
 FF05 5C01 A         SHL 0,1
 FF06 1201 A         BOC 2,.+2;                 BRANCH TO DATA RECORD,
 FF07 2123 A         JMP ENDREC;                    ELSE GO TO END RECORD.
 FF08 5CFF A         SHR 0,1
 FF09 3381 A         RCPY 0,3;                  RECORD BODY LENGTH IN AC3.
 FF0A 2924 A         JSR RDPCK
 FF0B 3181 A         RCPY 0,1;                  SAVE CHECKSUM.
 FF0C 4000 A         PUSH 0;
 FF0D 5101 A         CAI 1,1;                   AC1 HAS -(CKSUM MODE WORD).
 FF0E 290D A         JSR RDWDCK;                SLOUGH ADDRESS MODE.
 FF0F 290C A         JSR RDWDCK;                GET LOAD ADDRESS.
 FF10 3281 A         RCPY 0,2;                  LOAD ADDRESS IN AC2.
 FF11 290A A         JSR RDWDCK;                SLOUGH RELOCATION MODE WORDS.
 FF12 2909 A         JSR RDWDCK;
 FF13 3800 A         RADD 2,3
 FF14 5305 A         CAI 3,1+4;                 AC3 HAS -(LAST ADDRESS - 1).
 FF15 2906 A TTY2:   JSR RDWDCK;                GET DATA WORD.
 FF16 A200 A         ST 0,(2)
 FF17 4A01 A         AISZ 2,1;                  INCREMENT DESTINATION ADDRESS.
 FF18 3881 A         RCPY 2,0
 FF19 3C00 A         RADD 3,0
 FF1A 1204 A         BOC 2,TSTCKSUM;            IF DONE TEST CHECKSUM.
 FF1B 21F9 A         JMP TTY2
 FF1C 2912 A RDWDCK: JSR RDPCK
 FF1D 3100 A         RADD 0,1
 FF1E 0200 A         RTS
 FF1F 4400 A TSTCKSUM: PULL 0;                  GET CHECKSUM WORD.
```

```
REVISION-C 11/20/72                                     73227   09143181
CUTIL    CONTROL PANEL AND TTY UTILITIES                PAGE NUMBER    2
FF20 11DF A          BOC C1,ABSTTY
FF21 3481 A          RCPY 1,0
FF22 11DD A          BOC C1,ABSTTY
FF23 0000 A          HALT;                        CHECKSUM ERROR.
FF24 6117 A TORS:    AND 0,H3FFF;                 IGNORE TITLE AND SYMBOL RECORDS.
FF25 3181 A          RCPY 0,1
FF26 2908 A          JSR RDPCK
FF27 49FF A          AISZ 1,-1
FF28 21FD A          JMP .-2
FF29 2905 A          JSR RDPCK
FF2A 21D5 A          JMP ABSTTY
FF2B 2903 A ENDREC:  JSR RDPCK;                   SLOUGH CHECKSUM.
FF2C 2902 A          JSR RDPCK;                   SLOUGH ENTRY ADDRESS MODE.

                                                  GET ENTRY ADDRESS
FF2D 2901 A          JSR RDPCK;
FF2E 2000 A          JMP 0;                       JUMP TO PANEL ROUTINE.

FF2F                 .SPACE 3
FF2F 4100 A RDPCK:   PUSH 1
FF30 290C A          JSR RECV
FF31 5C08 A          SHL 0,8
FF32 4000 A          PUSH 0
FF33 2909 A          JSR RECV
FF34 4500 A          PULL 1
FF35 3482 A          RXOR 1,0
FF36 4500 A EXIT11:  PULL 1
FF37 0200 A          RTS
FF38 852E A DELAY:   LD 1,V2;                      DELAY SUBROUTINE
FF39 49FF A          AISZ 1,-1
FF3A 21FE A          JMP .-1
FF3B 0200 A          RTS
FF3C 3FFF A H3FFF:   .WORD X'3FFF


TELETYPE GET CHARACTER ROUTINE

FF3D                 .PAGE   'TELETYPE GET CHARACTER ROUTINE'
FF3D 000E A          JC14 = 14;                    INPUT JUMP CONDITION.
FF3D 0003 A          READR = 3;                    READER ENABLE FLAG.
FF3D 0004 A          XMIT = 4;                     TTY TRANSMIT FLAG.
FF3D 0001 A          C1 = 1
FF3D 0002 A          C2 = 2
FF3D 4100 A RECV:    PUSH 1;                       SAVE ACCUMULATORS.
FF3E 4200 A          PUSH 2
FF3F 0A80 A          PFLG 2;                       DISABLE LINK.
FF40 4E08 A          LI 2,8;                       SET COUNT FOR 8 BITS.
FF41 0800 A          SFLG READR
FF42 1E01 A          BOC JC14,.+2;                 TEST FOR START BIT.
FF43 21FE A          JMP .-1;                      LOOP UNTIL FOUND.
FF44 8521 A          LD 1,V1;                      LOAD TIMING PARAMETER.
FF45 29F3 A          JSR DELAY+1;                  DELAY HALF BIT TIME.
FF46 0B80 A          PFLG READR;
FF47 1E01 A          BOC JC14,.+2;                 TEST FOR DATA BIT.
FF48 21F6 A          JMP RECV+2
FF49 29EE A REP:     JSR DELAY;
FF4A 5CFF A          SHR 0,1;
FF4B 1E01 A          BOC JC14,.+2;
FF4C 6918 A          OR 0,H8000;
FF4D 4AFF A          AISZ 2,-1;                    DECREMENT COUNT.
FF4E 21FA A          JMP REP;
FF4F 29E8 A          JSR DELAY;
FF50 5CF8 A          SHR 0,8;
FF51 4600 A          PULL 2;
FF52 21E3 A          JMP EXIT11;                   THIS STEP DONE TO SAVE
FF53        ;                                       PROGRAM STORAGE SPACE.
```

(listing continued)

```
FF53                   .PAGE      'TRANSMIT CHARACTER ROUTINE'
FF53 4200 A SEND:      PUSH 2;                     SAVE ACCUMULATORS.
FF54 4100 A            PUSH 1;
FF55 0C00 A            SFLG XMIT;                   SEND START BIT.
FF56 29E1 A            JSR DELAY;
FF57 4E08 A            LI 2,8;                      SET BIT COUNT.
FF58 0C80 A PUT:       PFLG XMIT;
FF59 1301 A            BOC 3,$XX
FF5A 0C00 A            SFLG XMIT;
FF5B 290C A $XX:       JSR DELAY;                   SEND DATA BIT.
FF5C 5CFF A            SHR 0,1;
FF5D 4AFF A            AISZ 2,-1;                   TEST TO SEE IF DONE.
FF5E 21F9 A            JMP PUT
FF5F 0C80 A            PFLG XMIT;                   SEND TWO STOP BITS.
FF60 2907 A            JSR DELAY;
FF61 2906 A            JSR DELAY;
FF62 4500 A            PULL 1
FF63 4600 A            PULL 2
FF64 0200 A            RTS
FF65 8000 A H8000:     .WORD X'8000
FF66 0181 A V1:        .WORD X'0181
FF67 035E A V2:        .WORD X'035E


ABSPT

FF68                   .PAGE    'ABSPT'
FF68            ;
FF68            ;                 THIS IS A PAPER TAPE BOOTSTRAP ROUTINE THAT
FF68            ;                 READS 8 CHANNEL TAPE. THE FIRST 4 WORDS
FF68            ;                 DENOTE THE STARTING ADDRESS FOR THE OBJECT
FF68            ;                 PROGRAM BEING LOADED. THE LAST CHARACTER
FF68            ;                 ON THE TAPE MUST BE AN EXCLAMATION MARK.
FF68            ;
FF68            ;                 ENTRY POINT FOR THIS PROGRAM:  FF68
FF68            ;
FF68 4D04 A LOOP1:     LI 1,4
FF69 2906 A            JSR PTBOOT
FF6A 3881 A            RCPY 2,3;                READ FIRST 4 WORDS; THIS IS THE
FF6B            ;                               STARTING ADDRESS.
FF6B 4D04 A LOOP2:     LI 1,4;
FF6C 2903 A            JSR PTBOOT
FF6D A800 A            ST 2,(3);                READ AND STORE 4 HEX WORDS.
FF6E CD45 A            ADD 3,ONE;               INCREMENT MEMORY ADDRESS.
FF6F 21FB A            JMP LOOP2
FF70 29CC A PTBOOT:    JSR RECV;                GET ONE WORD.
FF71 6117 A PACK:      AND 0,MSKPAR;            MASK OUT PARITY BIT.
FF72 F11C A            SKNE 0,EXCLAM;
FF73 2141 A            JMP $OUT
FF74 F118 A            SKNE 0,CRETRN;           IGNORE  CARRIAGE RETURN.
FF75 21FA A            JMP PTBOOT
FF76 F117 A            SKNE 0,LINEFD;           IGNORE LINEFEED.
FF77 21F8 A            JMP PTBOOT
FF78 7112 A            SKAZ 0,NUMBER;
FF79 2105 A            JMP NUM
FF7A 7111 A            SKAZ 0,ALPHA;
FF7B 2102 A            JMP ALFA
FF7C 4C03 A            LI 0,3;                  ERROR CODE: INVALID CHARACTER.
FF7D 2000 A            JMP 0;
FF7E C109 A ALFA:      ADD 0,NINE;
FF7F 610A A NUM:       AND 0,MSK1
FF80 5E04 A            SHL 2,4
FF81 3282 A            RXOR 0,2
FF82 49FF A            AISZ 1,-1
FF83 21EC A            JMP PTBOOT
```

```
REVISION-C 11/20/72                                             73227    09143181
CUTIL     CONTROL PANEL AND TTY UTILITIES                       PAGE NUMBER     4

  182 FF84 0200 A           RTS
  183 FF85 4500 A $OUT:     PULL 1
  184 FF86 4500 A           PULL 1
  185 FF87 2131 A           JMP START;
  186 FF88 0009 A NINE:     .WORD 9
  187 FF89 007F A MSKPAR:   .WORD 07F
  188 FF8A 000F A MSK1:     .WORD 0F
  189 FF8B 0030 A NUMBER:   .WORD 030
  190 FF8C 0040 A ALPHA:    .WORD 040
  191 FF8D 000D A CRETRN:   .WORD 0D
  192 FF8E 000A A LINEFD:   .WORD 0A
  193 FF8F 0021 A EXCLAM:   .WORD 021


ASCII TAPE PUNCH ROUTINE

  194 FF90                  .PAGE      'ASCII TAPE PUNCH ROUTINE'
  195 FF90          ;
  196 FF90          ;                  THIS PROGRAM PUNCHES OUT ON PAPER TAPE
  197 FF90          ;                  THE CONTENTS OF A SPECIFIED RANGE OF
  198 FF90          ;                  MEMORY LOCATIONS. THE FIRST 4 WORDS ON
  199 FF90          ;                  THE TAPE  ARE THE ADDRESS OF THE STARTING
  200 FF90          ;                  LOCATION. THE LAST WORD ON THE TAPE IS
  201 FF90          ;                  AN EXCLAMATION MARK. PAPER TAPE GENERATED
  202 FF90          ;                  BY THIS ROUTINE MAY BE LOADED USING THE
  203 FF90          ;                  PTBOOT ROUTINE. BEFORE EXECUTING THIS
  204 FF90          ;                  PROGRAM, AC2 MUST BE LOADED WITH THE
  205 FF90          ;                  STARTING ADDRESS OF THE RANGE TO BE
  206 FF90          ;                  DUMPED AND AC3 MUST HAVE THE FINAL ADDRESS
  207 FF90          ;                  OF THE RANGE. EACH SET OF 4 WORDS ON THE
  208 FF90          ;                  TAPE ARE SEPARATED BY CARRIAGE RETURN
  209 FF90          ;                  AND LINEFEED CHARACTERS.
  210 FF90          ;
  211 FF90 3881 A PTPNCH:   RCPY 2,0;                   COPY STARTING ADDRESS INTO AC0.
  212 FF91 CD22 A           ADD 3,ONE;
  213 FF92 4D04 A ASC:      LI 1,4;                     COUNT FOR 4 HEX CHARACTERS.
  214 FF93 5804 A           ROL 0,4;
  215 FF94 4000 A           PUSH 0;
  216 FF95 61F4 A           AND 0,MSK1;
  217 FF96 E1F1 A           SKG 0,NINE;
  218 FF97 2112 A           JMP $NU
  219 FF98 01EF A           SUB 0,NINE;
  220 FF99 48C0 A           AISZ 0,-64
  221 FF9A 2988 A OUTT1:    JSR SEND;                   TRANSMIT CHARACTER.
  222 FF9B 49FF A           AISZ 1,-1;                  LOOP UNTIL 4 CHARACTERS SENT.
  223 FF9C 2108 A           JMP RETN
  224 FF9D 3C61 A           RCPY 3,0;                   CHECK TO SEE IF END OF
  225 FF9E 3882 A           RXOR 2,0;                       MEMORY RANGE.
  226 FF9F 110C A           BOC 1,$EXCL;                END OF DATA.
  227 FFA0 81EC A           LD 0,CRETRN;
  228 FFA1 29B1 A           JSR SEND;                   TRANSMIT CARRIAGE RETURN.
  229 FFA2 81EB A           LD 0,LINEFD;
  230 FFA3 29AF A           JSR SEND;                   TRANSMIT LINEFEED.
  231 FFA4 4400 A           PULL 0;                     POP TOP WORD TO CLEAR.
  232 FFA5 8200 A           LD 0,(2);                   BEGIN DATA TRANSMISSION.
  233 FFA6 C90D A           ADD 2,ONE;                  POINT TO NEXT LOCATION.
  234 FFA7 21EA A           JMP ASC;
  235 FFA8 4400 A RETN:     PULL 0
  236 FFA9 21E9 A           JMP ASC+1
  237 FFAA 48B0 A $NU:      AISZ 0,-80;
  238 FFAB 21EE A           JMP OUTT1
  239 FFAC 81E2 A $EXCL:    LD 0,EXCLAM;                TRANSMIT EXCLAMATION MARK.
  240 FFAD 29A5 A           JSR SEND;
  241 FFAE 2000 A           JMP 0;
```

(listing continued)

```
REVISION-C 11/20/72                                            73227    09143181
CUTIL     CONTROL PANEL AND TTY UTILITIES                      PAGE NUMBER    5
CONTROL PANEL ROUTINE

 242 FFAF                     .PAGE     'CONTROL PANEL ROUTINE'
 243 FFAF            ;        CUTIL CONTROL PANEL:    JULY 31, 1973.
 244 FFAF            ;
 245 FFAF            ;                                THIS CONTROL PANEL USES ALL DEVICE
 246 FFAF            ;                                ADDRESSES; THAT IS, NO ADDRESS
 247 FFAF            ;                                DECODING IS ASSUMED, AND THE PROGRAM
 248 FFAF            ;                                WILL RESPOND TO ALL 'RIN' AND 'ROUT'
 249 FFAF            ;                                INSTRUCTIONS.
 250 FFAF FFB1 A              .=X'FFB1
 251 FFB1 21B8 A JSTRT:       .WORD X'21B8
 252 FFB2 21EF A JINTR:       .WORD X'21EF
 253 FFB3 0005 A FIVE:        .WORD    5
 254 FFB4 0001 A ONE:         .WORD    1
 255 FFB5 8DFC A BEGIN:       LD 3,JINTR
 256 FFB6 AC01 A              ST 3,X'01;          LOAD LOCATION 1 WITH JUMP TO INTERRUPT
 257 FFB7 8DF9 A              LD 3,JSTRT;
 258 FFB8 AC00 A              ST 3,X'00;          LOAD LOCATION 0 WITH JUMP TO CONTROL PANEL
 259 FFB9 292F A START:       JSR SAVE       ;    SAVE ACCUMULATORS.
 260 FFBA 0900 A SET:         SFLG 1         ;    ENABLE INTERRUPT SYSTEM.
 261 FFBB 0600 A ROUT:        ROUT 0
 262 FFBC 1C04 A WAIT:        BOC C12,LA     ;    'LOAD ADDRESS' SWITCH.
 263 FFBD 1D0A A              BOC C13,LD     ;    'LOAD DATA' SWITCH.
 264 FFBE 1710 A              BOC C7,EX;          'EXECUTE' SWITCH.
 265 FFBF 1F25 A              BOC C15,DISP   ;    'DISPLAY' SWITCH.
 266 FFC0 21FB A              JMP .-4        ;    RETURN TO WAIT LOOP.
 267 FFC1 1CFF A LA:          BOC C12,LA     ;    CHECK RELEASE.
 268 FFC2 0400 A              RIN 0          ;    READ SWR.
 269 FFC3 3281 A              RCPY 0,2       ;    SAVE ADDRESS IN AC2.
 270 FFC4 3381 A              RCPY 0,3
 271 FFC5 E1ED A              SKG 0,FIVE;         PREVENTS LOADING OF RESERVED LOCATIONS
 272 FFC6 1213 A              BOC C2,RSRVE
 273 FFC7 21F3 A              JMP ROUT
 274 FFC8 1DFF A LD:          BOC C13,LD;         CHECK RELEASE FOR LOAD DATA SWITCH
 275 FFC9 0400 A              RIN 0;              READ SWITCHES
 276 FFCA F934 A              SKNE 2,LAST6;       PREVENTS LOADING LOCATION 6.
 277 FFCB 21F0 A              JMP WAIT
 278 FFCC A200 A              ST 0,(2)       ;    LOAD MEMORY
 279 FFCD C9E6 A              ADD 2,ONE;          INCREMENT ADDRESS
 280 FFCE 21EC A              JMP ROUT
 281 FFCF 17FF A EX:          BOC C7,EX      ;    CHECK RELEASE
 282 FFD0 0400 A              RIN 0
 283 FFD1 4000 A              PUSH 0         ;    SAVE JUMP ADDRESS IN STACK.
 284 FFD2 8406 A RSTOR:       LD 1,X'06
 285 FFD3 4100 A              PUSH 1
 286 FFD4 0280 A              PULLF
 287 FFD5 8002 A              LD 0,X'02      ;     RESTORE ACCUMULATORS.
 288 FFD6 8403 A              LD 1,X'03
 289 FFD7 8804 A              LD 2,X'04
 290 FFD8 8C05 A              LD 3,X'05
 291 FFD9 0200 A              RTS 0          ; FAKING AN INDIRECT JUMP.
 292 FFDA 0600 A RSRVE:       ROUT 0
 293 FFDB 1D05 A              BOC C13,LDAC
 294 FFDC 1F01 A              BOC C15,DISPAC
 295 FFDD 21FD A              JMP .-2
 296 FFDE 1FFF A DISPAC:      BOC C15,DISPAC;      DISPLAY ACCUMULATOR ROUTINE
 297 FFDF 8302 A              LD 0,2(3)
 298 FFE0 21DA A              JMP ROUT
 299 FFE1 1DFF A LDAC:        BOC C13,LDAC
 300 FFE2 0400 A              RIN 0
 301 FFE3 A202 A              ST 0,2(2)
 302 FFE4 21D6 A              JMP ROUT
 303 FFE5 1FFF A DISP:        BOC C15,DISP
 304 FFE6 8300 A              LD 0,(3)
 305 FFE7 CDCC A              ADD 3,ONE;          INCREMENT ADDRESS
```

```
REVISION-C 11/20/72                                            73227   09143181
CUTIL    CONTROL PANEL AND TTY UTILITIES                       PAGE NUMBER     6
CONTROL PANEL ROUTINE
  306 FFE8 2102 A          JMP ROUT
  307 FFE9            ;
  308 FFE9 A002 A SAVE:    ST 0,X'02     ;  SAVE AC0 - AC3 IN LOCATIONS X'02 - X'05.
  309 FFEA A403 A          ST 1,X'03
  310 FFEB A804 A          ST 2,X'04
  311 FFEC AC05 A          ST 3,X'05
  312 FFED 0080 A          PUSHF
  313 FFEE 4500 A          PULL 1
  314 FFEF A406 A          ST 1,X'06;       SAVE FLAGS IN LOCATION 6.
  315 FFF0 0200 A          RTS 0
  316 FFF1            ;


INTERRUPT SERVICE ROUTINE FOR HALT AND STACKFULL

  317 FFF1                 .PAGE        'INTERRUPT SERVICE ROUTINE FOR HALT AND STACKFULL'
  318 FFF1            ;
  319 FFF1 4CFF A INTR:    LI 0,-1;
  320 FFF2 1803 A          BOC C8,STFL;
  321 FFF3 29F5 A          JSR SAVE;    SAVE ACCUMULATORS IN LOCATIONS  2,3,4 AND 5.
  322 FFF4 4400 A          PULL 0
  323 FFF5 4000 A          PUSH 0
  324 FFF6 0600 A STFL:    ROUT 0;
  325 FFF7 0000 A ZERO:    HALT;
  326 FFF8 0400 A          RIN 0;       READ SWITCHES
  327 FFF9 1102 A          BOC C1,.+3;  IF SWITCHES ARE SET TO ALL ZEROS, 'EXECUTE' CAUSES
  328 FFFA 29D7 A          JSR RSTOR;   A RETURN TO THE CONTROL PANEL ROUTINE.
  329 FFFB 0100 A          RTI ;        IF SWITCHES ARE SET TO ANY NON-ZERO NUMBER,'EXECUTE
  330 FFFC            ;                 CAUSES A NORMAL RETURN FROM INTERRUPT.
  331 FFFC 4400 A          PULL 0
  332 FFFD 21BC A          JMP SET;     RETURN TO CONTROL PANEL ROUTINE.
  333 FFFE 2186 A          JMP BEGIN
  334 FFFF 0006 A LAST6:   .WORD 6
  335  000                 .END

   ******      0 ERRORS IN ASSEMBLY   ******
```

(listing continued)

```
REVISION-C 11/20/72                                                 73227   09143181
CUTIL    CONTROL PANEL AND TTY UTILITIES                            PAGE NUMBER    7
INTERRUPT SERVICE ROUTINE FOR HALT AND STACKFULL

$EXCL   $NO     $OUT    $XX     ABSTTY ALFA    ALPHA   ASC     BEGIN   C1
FFAC A  FFAA A  FF85 A  FF5B A  FF00 A FF7E A  FF8C A  FF92 A  FFB5 A  0001 A

C11     C12     C13     C14     C15    C2      C3      C4      C5      C7
000B A  000C A  000D A  000E A  000F A 0002 A  0003 A  0004 A  0005 A  0007 A

C8      CRETRN DELAY    DISP    DISPAC ENDREC  EX      EXCLAM EXIT11 FIVE
0008 A  FF8D A  FF38 A  FFE5 A  FFDE A FF2B A  FFCF A  FF8F A  FF36 A  FF83 A

H3FFF   H8000   INTR    JC14    JINTR  JSTRT   LA      LAST6   LD      LDAC
FF3C A  FF65 A  FFF1 A  000E A  FFB2 A FFB1 A  FFC1 A  FFFF A  FFC8 A  FFE1 A

LINEFD LOOP1   LOOP2   MSK1    MSKPAR NINE    NUM     NUMBER ONE     OUTT1
FF8E A  FF68 A  FF6B A  FF8A A  FF89 A FF88 A  FF7F A  FF8B A  FF84 A  FF9A A

PACK    PTBOOT PTPNCH PUT     RDPCK  RDWDCK READR   RECV    REP     RETN
FF71 A  FF70 A  FF90 A  FF58 A  FF2F A FF1C A  0003 A  FF3D A  FF49 A  FFA8 A

ROUT    RSRVE   RSTOR   SAVE    SEND   SET     START   STFL    TORS    TSTCKS
FF8B A  FFDA A  FFD2 A  FFE9 A  FF53 A FFBA A  FFB9 A  FFF6 A  FF24 A  FF1F A

TTY1    TTY2    V1      V2      WAIT   XMIT    ZERO
FF03 A  FF15 A  FF66 A  FF67 A  FFBC A 0004 A  FFF7 A

   87B8     A617
```

# Notes

# Notes

# Notes

# Notes

# Notes

# Notes

FHWA
R&D